AN INPUT/OUTPUT TERMINAL FOR INDIAN LANGUAGES

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By
ANIL K. PATHAK

BBAB

to the
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
AUGUST, 1978

I.I.T. KANPUR
NIKAL LIBRARY
No. 55455

OCT 1978

EE-1978-M-PAT-INP

S.870

CERTIFICATE

Certified that this thesis work entitled 'A MODULAR INPUT/OUTPUT TERMINAL FOR INDIAN LANGUAGES' has been carried out under my supervision by Mr. A.K. Pathak, and that it has not been submitted elsewhere for a degree.

(Dr. R.M.K. Sinha)
Assistant Professor
Dept. of Electrical Engineering
and
Computer Science
Indian Institute of Technology
Kanpur 208016

ACKNOWLEDGEMENT

I hereby take the much wanted opportunity to thank my guide Dr. R.M.K. Sinha for suggesting to me the excellent topic for my thesis and giving me full freedom of action and thought interspersed with useful discussions.

Thanks is not enough for Mr. Arjun Raman, Research Engineer, ACES. I owe him this project.

Thanks are also registered herewith to my project partners Mr. M.P. Sastri, K.P. Laturkar and B.V. Ramana for the fruitful discussions I had with them.

I must also thank all my friends, who have rendered my stay in this Institute pleasant and enjoyable. S.S. Pujari deserves extra thanks for his immense help in drawing the figures.

Credit for typing goes to Mr. R.K. Jain of Computer Centre.

August, 1978 — A.K. Pathak

CONTENTS

ABSTRACT

The present work forms a part of the project to develop a universal stand-alone, modular I/O terminal for Indian languages. The system modules of a phonetic based keyboard, CRT dot matrix display composition processor and system controller have been clearly identified and interfaced over the IEEE-488 bus, in accordance with IEEE std. 488 (1975 (specifications for interfacing programmable digital instruments), to form the complete I/O terminal . The functions of the system controller in bus management and execution of the text editing routines have been specified and implemented on a 6800 microcomputer.

CHAPTER I

INTRODUCTION

The thirty odd years after independence have witnessed an ever increasing use of almost all of the sixteen major Indian languages in various official and academic activities of Center and states. Their wide spread use in government offices, business houses, educational institutions and printin industry is only too well known. Infact, the amount of official and academic activity going on in regional languages is so large that an immediate need is felt for developing devices which can handle and process data directly in these languages, thus bringing automation and speed into their use. Latest technological advances in areas like LSIs, micro-processors, high resolution CRTs, dot matrix printers etc. have opened new vistas in the development of such peripherals and terms like intelligent typewriters; communication modems for teleprinter and telex systems; computer based data storage and retrieval systems such as data banks and translation/ transliteration facilities; and computer based phototype setting and photocomposition systems for high quality printing in Indian scripts are increasingly being talked about [1,2].

Early attempts in this direction [7] relied heavily on the existing technology for Roman script. While some of these methods still continue to be popular, or atleast being

explored, they present many difficulties in implementation because of the inherently complex nature of Indian scripts, made so due to the presence of half characters, vowel modifiers and diacritical marks, apart from the basic symbols, in the text and the complex and often contaxt dependent way of joining them to form composite characters [3]. Several compromises must then be made at the cost of quality.

Narsimham in 1971 and Sinha in 1973 [4 and 5 respt.] independently proposed schemes for the mechanization of Indian scripts which deviated significantly from the existing philosophy. Since most of the Indian scripts, they argued, have their origin in Bramhi script, and hence are similar in structure, it should be possible to develop peripherals which are universely adaptable to all Indian languages. Phonetic based keyboarding schemes (basic to all subsequent data processing) universely applicable to all Indian languages belonging to Bramhi family were also proposed. To explore the practicability of the philosophy a project to develop a stand alone I/O terminal for Indian languages was jointly started at Electrical Engineering and Computer Science Departments of I.I.T. Kanpur. The present work forms a part of the project and relates mainly to interfacing various independently developed modules of the system, namely, the keyboard, composition processor, CRT display and the system controller along

the lines described by Roman and Pathak [16,17].

1.1.1 What is an Interface

In most general terms interface is the 'totality of means adopted to bring coordination between two (or more) independent processes' [6]. Thus we have a repertoire of interfaces embracing all conceivable branches of science and arts - from man-machine interfaces like console switches, programming languages etc. to mechanical interfaces like connecting rods, gears etc. Some say that language is also a form of interface to communicate independent thought processes of two individuals.

Basically, an interface attempts at removing the incompatibilities on the two sides making their 'interconnection' possible.

In digital instruments the incompatibilities could be many - different word lengths and data rates, different logic levels, inconsistent word formats and codes etc. Interfacing then would imply the hardware and software means adopted to permit the interconnection of two or more of such devices making them function as a single integrated system.

1.1.2 The Standard Interface: Historical Evolution

Rapid advances in the field of digital instrumentation left some worst sufferers especially the interface designers. They were faced with the task of interfacing progressively more complex modules, and, more importantly, every restructuring of the existing system (say replacing an obsolete module with new one) or its expansion meant a complete overhauling of the interface system. The hardware and/or software costs and time lags involved to do so were often formidable and hence discouraging. Look out, therefore, began for an interface structure which would be an international standard, bringing in long cherished features of flexibility cost effectiveness and speed in the interface design.

A number of localized standards soon started emerging and some of them like CAMAC [6] gained wide popularity in data acquisition systems of satellite telemetery and nuclear reactor monitoring. Most of such 'standard' interfaces, however, catered only to some specific group of applications and did not gain much popularity in general.

In 1972 Hewlett Packard Company of USA proposed an interface bus structure for digital data based on byte serial bit parallel asynchronous data transfer between the interconnected devices. This scheme with some modifications was accepted by IEEE in 1975 and came to be known as IEEE std 488 bus or HP interface bus (HP-IB). Later the same bus structure was also

adopted by IEC and ANSI and was formally given the status of an international standard. The main advantages promised by the standard interface bus are as follows:

1. Compatibility of instruments manufactured by different manufacturers - since all of them will be IEEE-488 bus compatible.
2. Modularity and flexibility of integrated systems.
3. Cost effectiveness.

The ease of interface design is now obvious. Since all devices must be HP-IB compatible, the interface hardware design is straightforward and can be taken up at the design stage of the device. Only the necessary software need now be written for making a complete interface design. (Presence of atleast one computing device, which will execute the software routines, is assumed in the system). Flexibility results from the ease with which different routines can be called for different system configurations.

1.2 IEEE Standard Digital Interface for Programmable Instrumentation (1975). A Brief Introduction

In following passages, a brief introduction to the relevant features of the IEEE std. 488 [9,18,19] are given. Detailed specifications for interface systems designed around the IEEE-488 bus may be found in reference [8].

1.2.1 Bus structure

Any effective communication among a group of interconnected devices requires the presence of following three basic functional elements.

(i) A talker (device) to send device data over the interface bus.

(ii) One (or more) listener (devices) to receive the device data being sent over the bus.

(iii) A controller (device) to manage and schedule the data flow over the bus, primarily by designating which devices are to act as talker or listener during each data transfer sequence and subsequently by scheduling the data transfer.

All such communications within a system take place over the standard interface bus shown (Fig. 1.1) in a typical system configuration. All the devices are programmable in that they can be programmed to perform any of the several device and interface functions built into them.

The bus contains sixteen signal lines grouped into following three sets:

(i) Eight bidirectional data lines to carry coded messages such as device data, device programming data, status bytes, device addresses and control commands.

(ii) Three lines for sending handshake signals NRFD, NDAC and DAV for an unambiguous, asynchronous data transfer

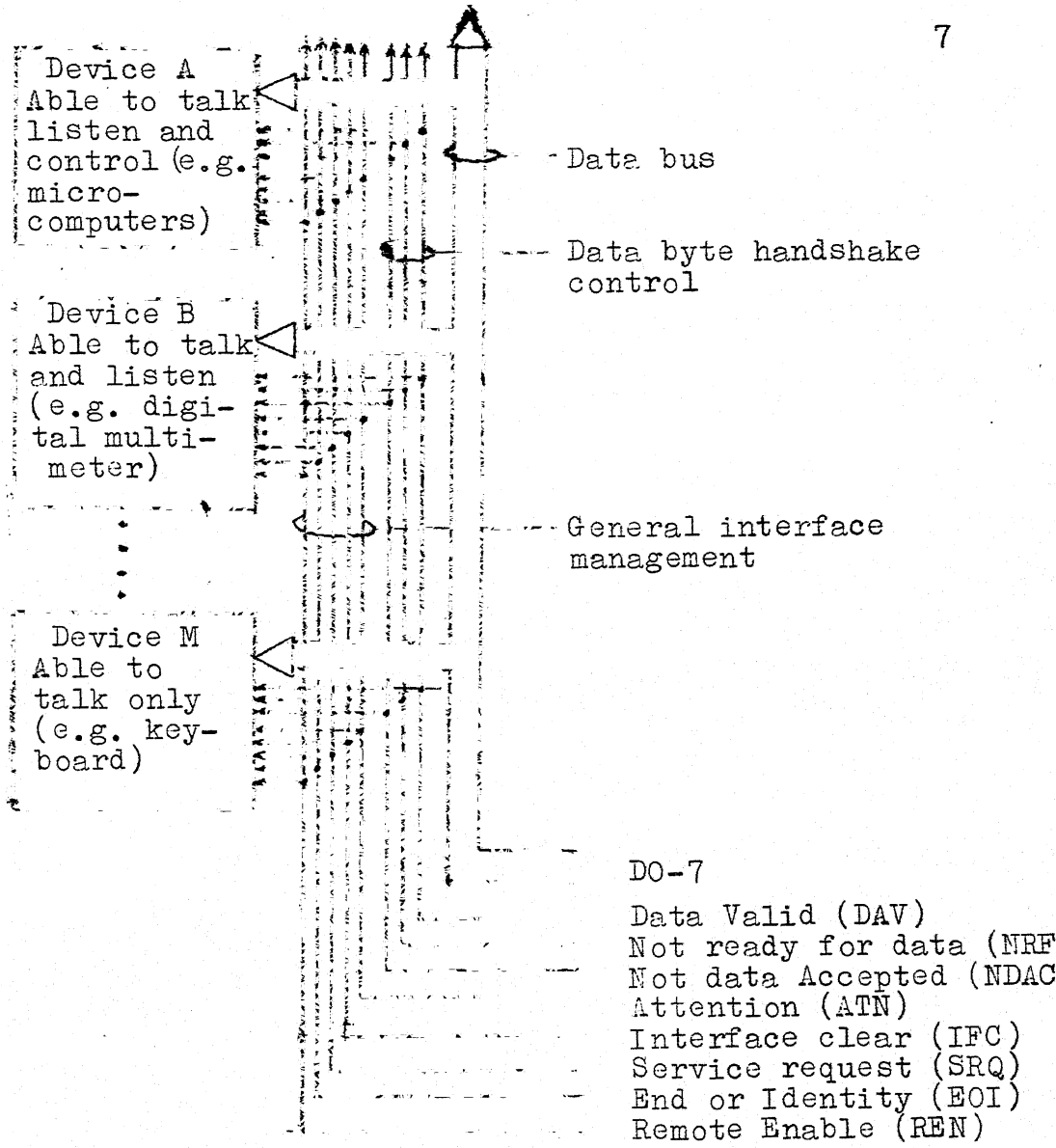


Fig. 1.1: IEEE-488 bus in a typical system configuration

over the bus. The data transfer cycle is initiated by all 'active' listeners of the system sending a 'Hi' on NRFD line indicating their readiness to receive the data byte (negative logic : Hi = False, Lo = True). The active talker responds by placing the byte on the bus and sending a 'lo' on DAV line. When all listeners have successfully received the byte, a 'Hi' on NDAC line is sensed by the talker, which then removes

the DAV signal completing the handshake cycle. Data rate is thus adjusted to the slowest listener participating in the transfer sequence.

(iii) The remaining five lines are used for general interface management:

ATN line is used by the active controller to indicate a controller take over of the bus. Thus, when ATN is 'lo', all devices must receive the messages being sent over the bus by controller and interpret them as control commands or device addresses. An appropriate action by them must follow.

IFC line is used by controller to place the interface system in a known quiescent state.

Service requests from all devices having capability to interrupt the controller are ORed into one SRQ line which finally goes to the controller.

EOI line may be used, with ATN false, by the current talker to indicate end of a message string.

1.2.2 Interface functions

An interface function is a functional element which provides a device with the capability to send, receive (and/or process) data over the bus. Hence a talker interface function, mentioned earlier, when 'active' within a device, sends data over the bus to other devices acting as listeners. IEEE std. 488 specifies a set of ten interface functions to

aid in this data transfer process over the bus. Apart from the interface functions of talker, listener and controller mentioned earlier, the standard set includes following seven interface functions: Source handshake, Acceptor handshake, Service request, Parallel poll, Device clear, Remote local and Device trigger. Although all of these may not be necessary in a particular device, each one of them can find uses in a typical system configuration. The interface designer has the freedom to select any combination of them for a particular device to give it the required interface capabilities. A keyboard, for example, is given interface functions of talker, source handshake and service request only since it can only 'talk' and 'interrupt' (when a new key is depressed).

All the interface functions are described by a strictly defined set of interlocked state sequences sending out well defined interface messages over the bus (Interface messages, as distinct from device data, are meant to cause state transitions within other interface functions connected to the bus and are sent, broadly, to manage the bus). The state diagrams of interface functions and interface messages to be sent while in a particular state are described in Chapter 2.

1.2.3 Functions of the controller

The controller interface function attached to a computing device of the system executes several software routines to

manage the bus. Its main functions are briefly listed below:

(i) It designates devices connected to the bus as talkers or listeners by sending out their respective talk or listen addresses at the start of every new data transfer sequence.

(ii) It acknowledges service requests from different devices by executing a serial poll routine wherein it sequentially tests the status of each 'potential' interrupter and having found one, it goes about executing the corresponding service request routine of the device. The conflicts arising from simultaneous interrupts from several devices are resolved by a pre-established priority among the devices.

(iii) At power on, it initializes the system by sending IFC signal, and/or, in association with device clear and device trigger interface functions, by sending out device clear signals.

1.3 Definition of the Problem

A typical stand alone I/O terminal for Indian languages consists of the following devices:

1. Keyboard for inputtting the data/text. An 8 bit code corresponding to the key depressed is generated by the device.
2. A CRT display (and/or matrix printer) for displaying the data/text as it is inputted from the keyboard, or, as an output device, for displaying the data outputted by a

computer or a communication channel.

3. A composition processor which accepts the code from the keyboard and generates such information in proper format which will be used by the CRT display to display the inputted character.

A fourth device called system controller was found desirable to be introduced into the system for doing the controller/bus management jobs in addition to executing various text editing and other special purpose routines associated with the keyboarding of the text. While these jobs could well have been entrusted to the composition processor as well, immediate system considerations (non availability of the composition processor at present, for one) favoured the treatment of the system controller as a separate device. A little advantage is however to be gained from the present setup because the system tasks of bus scheduling and character composition have been isolated into two parallel processes, and system efficiency is higher. With the future expansions of the basic I/O system very much in the offing the bus scheduling tasks shall become more and more elaborate and justify the separate identity of the system controller.

In their paper Raman et.al [1] have emphasized the need for a systematic development of a compatible set of modular peripherals in Indian Languages which when

interconnected appropriately in different configurations, would yield any one of the variety of systems (referred to in the first paragraph) - thus optimising the overall development effort in this direction. The adoption of the standard IEEE 488 bus structure for the system configurations is thus immediately advocated.

Our problem can thus be briefly stated as: to interface the four system modules - over an IEEE-488 bus and to configure them into an intelligent stand alone I/O terminal. A handy 6800 microcomputer was picked up for the system controller and the keyboard was separately designed and fabricated during the course of the project.

1.4.1 Basic System Configuration

The block diagram of the system configuration is shown in Fig. 1.2. The system modules of phonetic based keyboard with 2-key roll over facility, composition processor, CRT dot matrix display and system controller have been interfaced to the IEEE-488 bus through the hardware implementations of requisite interface functions plus device dependent local interfaces, if necessary. Following interface capabilities have been given to the different modules of the system.

(i) Keyboard: Talker and Source handshake to send status byte and key code; Acceptor handshake to receive control commands and service request.

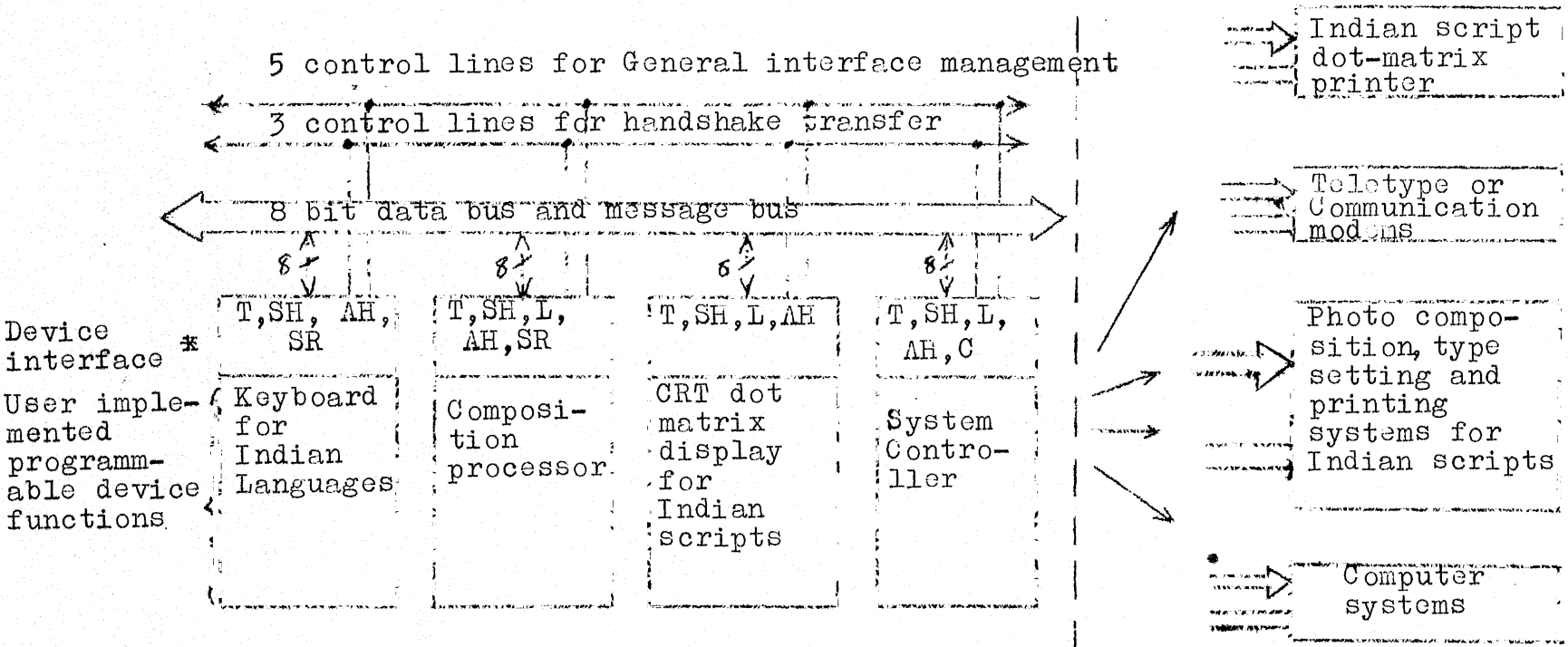


Fig. 1.2: Basic configuration of the stand-alone I/O terminal for Indian scripts

* IEEE std. 488-1975 defined interface functions + local interface if necessary.

** The system shown to the left of dotted lines can be easily extended/reconfigured to suit anyone of the applications shown to the right.

(ii) Composition processor: Talker and Source handshake to send status bytes and the composed characters; Listener and Acceptor handshake to receive the key code and control commands; and the service request.

(iii)CRT dot matrix display: Talker and source handshake to send its memory contents to system controller during edit routines; Listener and acceptor handshake to receive the composed character from composition processor and the programming bytes from system controller.

(iv) System controller: Talker, controller and source handshake to send control commands, device addresses etc. and modified display memory contents back to the display during various editing and general system management routine; Listener and Acceptor handshake to receive status bytes and display memory contents during editing routines.

At power on the system controller initializes the system and goes to a standby state waiting for a service request to come. The service request can come either from keyboard when a new key is pressed or from composition processor when a new composite character is ready. Having received the service request it executes a serial polling routine to locate the requesting device.

The status byte sent by the device to the system controller contains the following information:

executed no more service requests are entertained.

1.4.2. System efficiency

Assuming the maximum keyboarding speed, because of physical limitations, of 5 key depressions per second ($\simeq$ 60 wpm) the system has something like 200 m secs wherein to process one character. However to achieve this maximum physical keyboarding speed there should an undiscernible delay between the depressing of the key and appearance of the corresponding character on the screen. Assuming the maximum allowable delay of 50 msec (1/20 sec), the processing of one character must be complete within this limit.

The processing of a character broadly takes the following steps:

(i) Recognition of a depressed key by the keyboard electronics and subsequent sending of a service request.

(ii) Recognition of the service request by the system controller and establishing the desired communication path.

(iii) Reception of the key code by the composition processor and generation of display compatible information corresponding to the key depressed.

(iv) Repeat step (ii)

(v) Reception of the information by the CRT display and subsequent display of the character.

While these steps will be dealt with in some detail at

appropriate points in related chapters to come, it is worth mentioning here that most of this processing time will be taken by step 3.

In the remaining work Chapter 2 discusses the hardware implementation of the state diagrams of interface functions and the system aspects of display and composition processor interfacing. Chapter 3 describes a phonetic based keyboard with two key roll over facility and its interfacing to the IEEE-488 bus. Chapter 4 explains the software implementation of the state diagrams of interface functions on the 6800 microcomputer and details the varied functions of the system controller in system management.

CHAPTER II

DEVICE INTERFACING

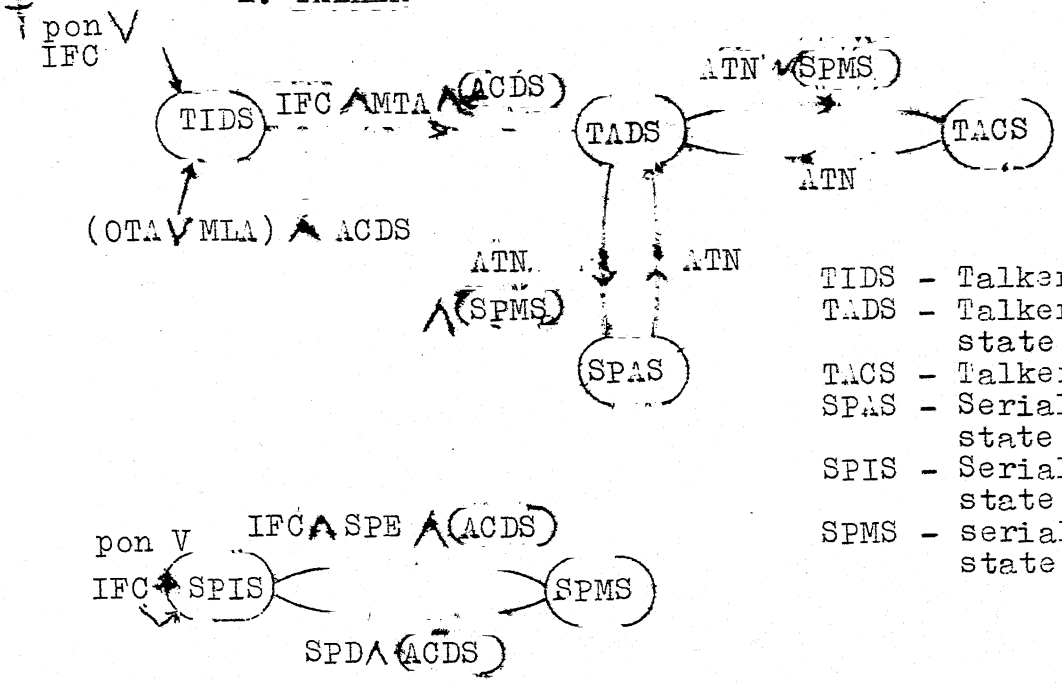
The general design strategy to be adopted in the interface system design involves interfacing all the system modules to a common unified IEEE-bus and various communications among the modules connected to the bus according to a fixed protocol. The description of hardware realizations of requisite interface functions must thus be the first step towards general interface description.

2.1.1 Interface functions: state diagrams

All interface functions of the IEEE standard have been defined in terms of groups of interlocked states [8]. At any instant only one state can be active within a group and transitions within a group occur, when the corresponding conditions for transition have been met. Since a device can have more than one interface functions, the total capability of the device in that regard at any instant is equal to the logical conjunction of the active states of all its interface functions.

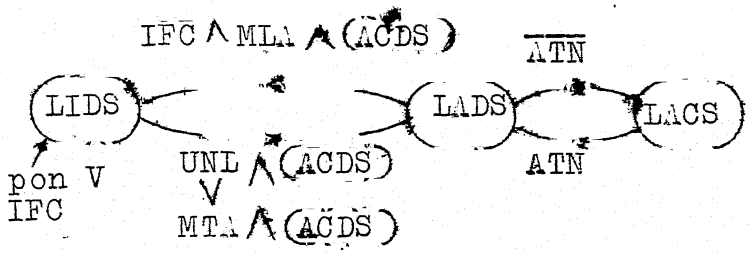
The state diagrams of the relevant interface functions are reproduced in Fig. 2.1 from reference [8] for clarity before their implementation is attempted. Some of the state diagrams have been slightly modified to suit their hardware implementations.

1. TALKER



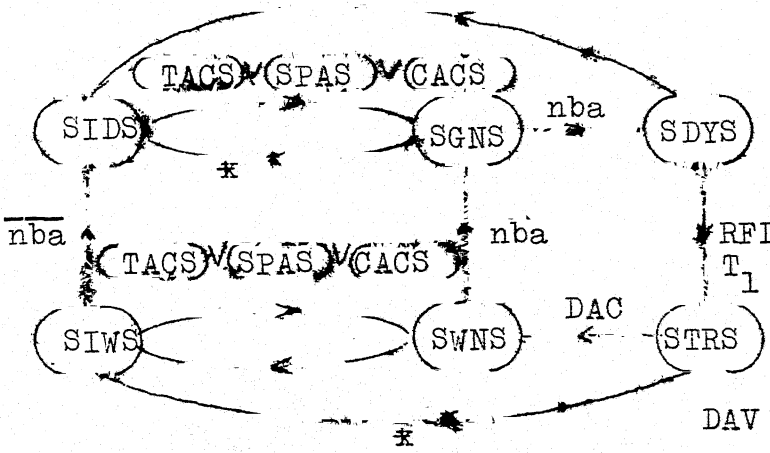
TIDS - Talker idle state
TADS - Talker addressed state
TACS - Talker active stat
SPAS - Serial poll active state
SPIS - Serial poll idle state
SPMS - serial poll mode state

2. Listener



LDIS - Listener idle state
LADS - Listener addressed state
LACS - Listener active state

3. Source handshake



SIDS - Source idle state
SGNS - Source generate state
SDVS - Source delay state
STRS - Source transfer state
SWNS - Source wait for new cycle state
SIWS - Source idle wait state

* = (ATN ∧ (CACS) ∧ (CTRS)) ∨ (ATN ∧ (TACS) ∧ (SPAS))

† This, and other acronyms have been explained in Appendix B

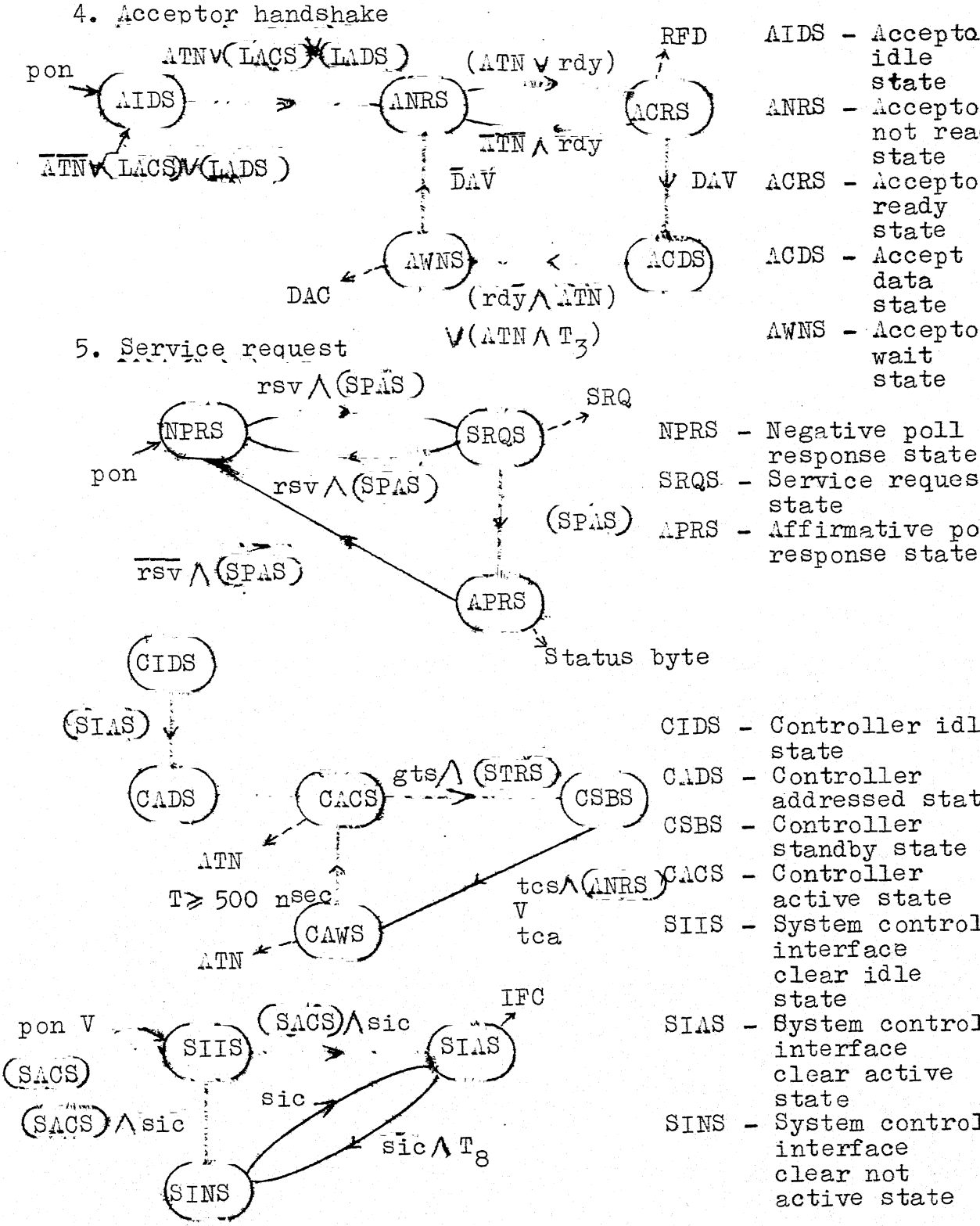
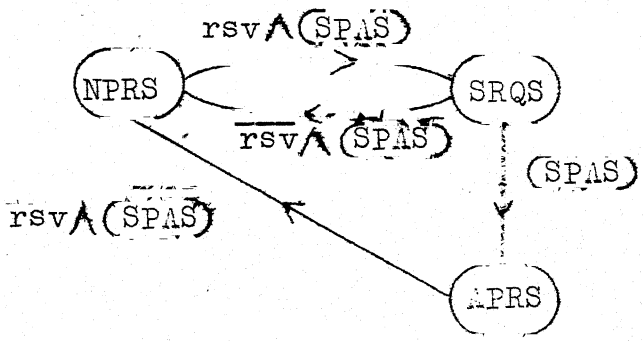


Fig. 2.1: State diagrams of interface functions as described in IEEE std. 488-1975.

2.1.2 Hardware Implementation [10]:

Two distinct approaches were tried for the hardware implementation of state diagrams.

2.1.2.1 The sequential circuit approach: This approach is best illustrated by implementing the SR interface function. Negative logic has been used throughout. The state diagram of SR function is reproduced below for visual clarity.



If arbitrary codes of NPRS-00, SQRS-01 and APRS-11 are assigned to the three states, the following state transition table is obtained

	Present state		Next state			
	Q_1	Q_o	RS* = 00	01	11	10
NPRS	0	0	00	01	00	00
SQRS	0	1	11	01	00	11
APRS	1	1	11	11	00	11
-	1	0	XX	XX	XX	XX

* R = rsv, S = SPAS

Simplification by K-maps gives the following expressions for D_o and D_1.

$$D_o = Q_o \cdot \overline{(SPAS)} + rsv \cdot \overline{(SPAS)}$$

$$D_1 = Q_o \cdot \overline{(SPAS)} + Q_1 \cdot rsv$$

The following sequential circuit is thus the hardware implementation of SR function.

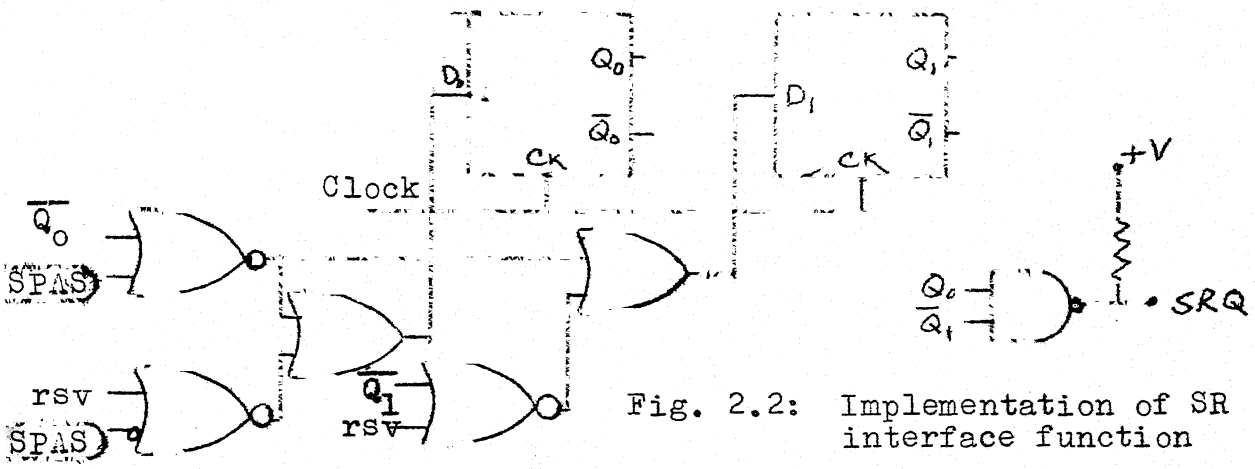


Fig. 2.2: Implementation of SR interface function

It is obvious that this approach is unsuitable for most other interface functions because of the very large number of transition parameters involved. A second modular approach which bypasses this hurdle elegantly is now discussed.

2.1.2.2 The Modular approach: A careful study of the state diagrams reveals that, while in one state, the interface functions continuously check for either of upto two conditions for transition being true and in the event of either of them becoming true, make transition to the

appropriate next state. This rather trivial fact is exploited in the modular approach discussed below.

Fig. 2.3 shows the simplified block diagram of the scheme.

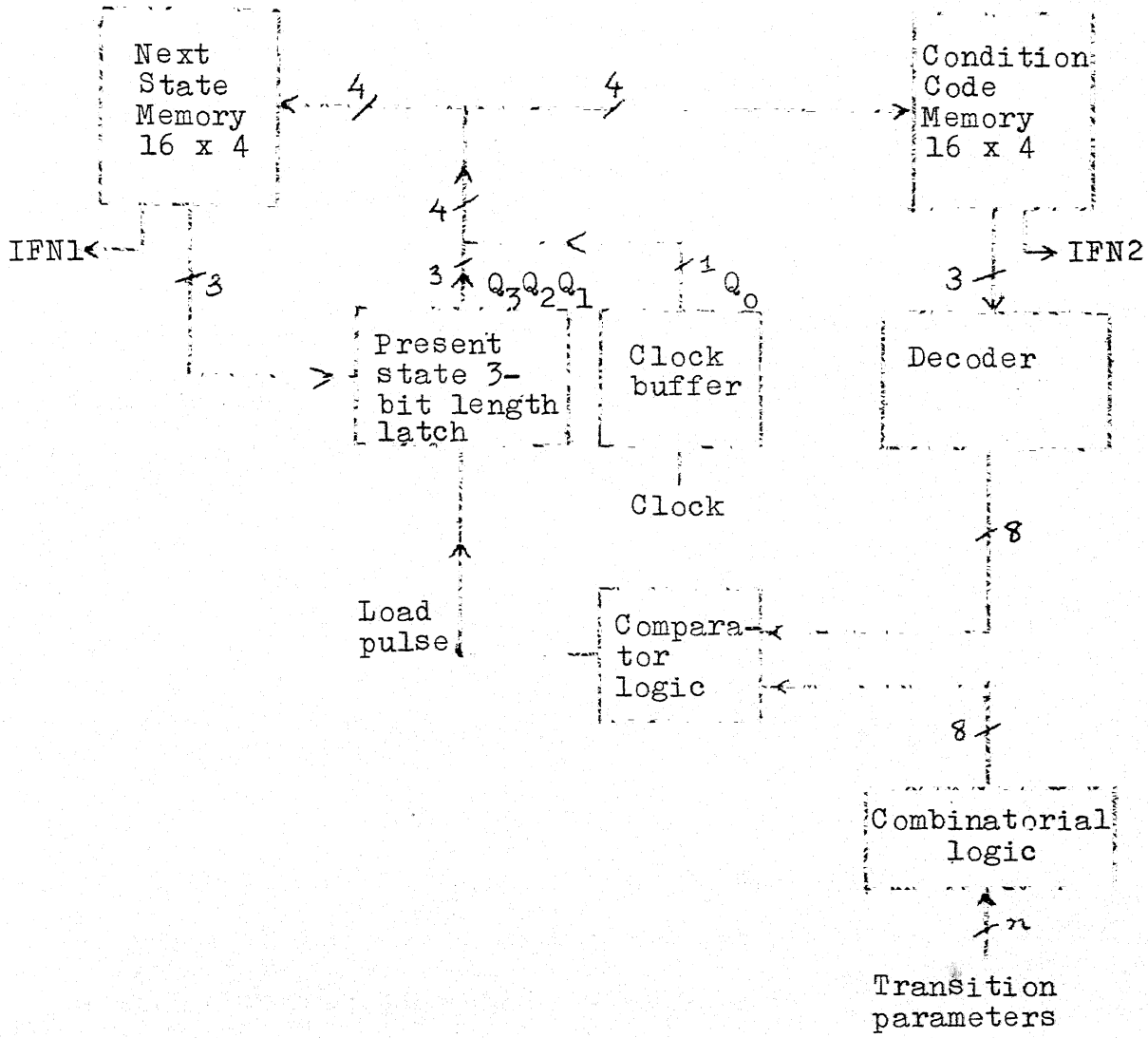


Fig. 2.3: Hardware realization scheme for interface functions based on the modular approach

Any explanation of the functioning of the schematic must start from the present state [PS] latch which contains at any instant the current active state of the interface function. The three bit output of the latch when combined with the clock buffer output generates two 4 bit addresses $Q_3Q_2Q_1Q_o$ (Q_o = 0,1) which are presented simultaneously to a Next State (NS) memory and a condition code (CC) memory both. Corresponding to the two addresses presented to it the CC memory presents the codes corresponding to the two transition conditions to be checked for to the 3-8 line decoder. Similarly the NS memory presents the two next states, from the present active state, to the PS latch synchronously with the corresponding conditions being presented to the decoder by CC memory. The continuously running clock ensures that the two conditions are consistently being checked, one after the other, until one of them become true.

The individual transition parameters (eg. nba, TACS etc.) are combined by logic hardware into the separate transition conditions for the interface function and the status about each of these conditions is consistently presented to the comparator logic. (The design assumes that a maximum of 8 transition conditions will be called upon to be tested in an interface function). The decoder output at any instant specifies, depending on the condition code presented to it by the

CC memory, which of the condition must be tested by the comparator. When any condition is met, the comparator gives out a load pulse to the PS latch and the next state being presented to it by the NS memory is latched into it. The cycle is thereafter repeated for the new active state. The fourth unused bits of the NS and CC memories are used to generate such interface messages which must be generated while one interface state is active. Examples include NRFD, NDAC, ATN etc.

The main advantage of this scheme, besides others, is its modularity. It is obvious that implementations of different interface functions with this scheme will differ only in the combinatorial logic for transition conditions and the programming of NS and CC memories. This feature has been fully utilized in fabricating and testing Talker, Listener, Source handshake, Acceptor handshake and controller interface functions.

Conventional RAMs are used for the NS and CC memories and their programming is undertaken by the system controller during the system initialization routine.

While a number of modifications can immediately spring to mind (like using PLA's in place of RAM's for NS and CC memories [13]) each one of these must be considered in view of the application to which the interface is being put. For example, in our system, (and for reasons to become clear later) the interface efficiency was one of the important d sign considerations. Since setting up of interface protocol through interface function

state transitions was feared to take away a considerable portion of the total intercommunication time, it was desirable to run it at a maximum clock rate. The TTL memories were thus favoured to MOS PLA's. A clock rate of 2.5 MHz was successfully achived.

2.1.3 The Interface Function Repertoire Units (IFR Units)

The hardware implementations of the interface functions being device independent have been done for the Talker, Listener, SH, AH, SR, controller and device trigger interface functions and assembled into standard IFR modules in conjunction with ACES, IIT, Kanpur. These units are intended to be used for interconnecting non-bus-compatible programmable instruments into integrated systems for laboratory purposes. In a typical application a non-controller device is interfaced to the IEEE-488 bus in the following manner.

Since the IEEE-488 bus is a relatively recent phenomenon in international market, therefore, while new instruments are being designed to be IEEE-bus compatible, the existing instruments are being hastily modified to be HP-IB compatible. A local interface, apart from the hardware implementations of requisite interface functions, becomes necessary in these instruments, which essentially does the following jobs:

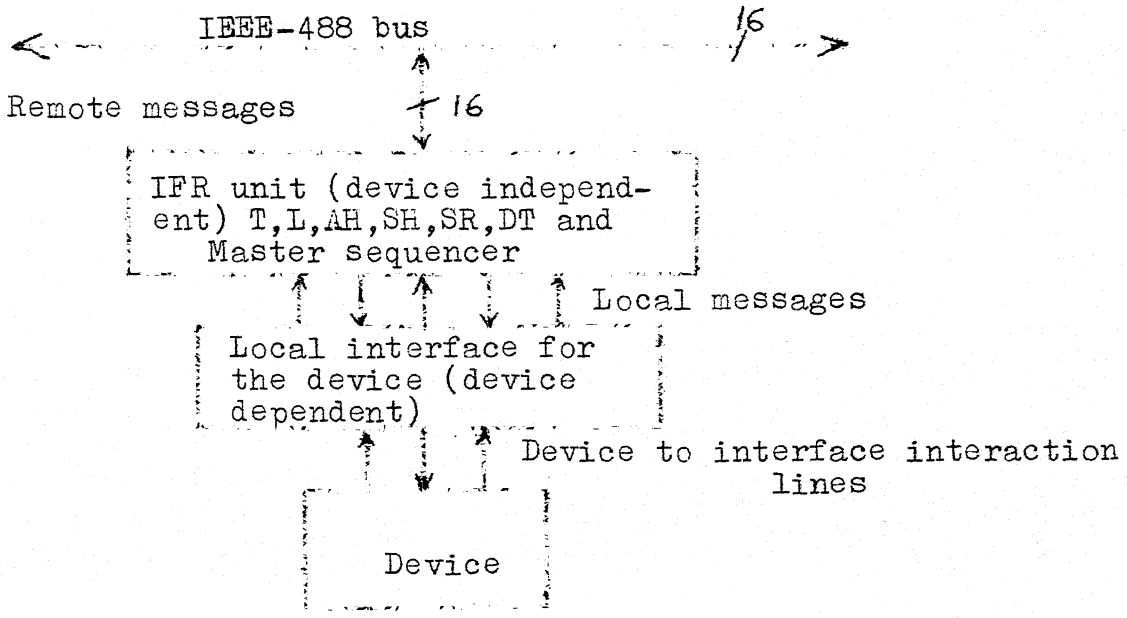


Fig. 2.4: A typical programmable device to IEEE-bus interfacing

(i) Interacts with the device and IFR unit to generate such local messages like rdy, nba etc, specified in the IEEE standard.

(ii) If the internal word length of the device is more than 8, it slices the word into 8 bit bytes and loads them serially on the interface data bus while talking to other devices connected to the bus. While listening it does the reverse process of concatenating the bytes into full length word of the device. The most significant byte of the word is sent first while talking.

(iii) If the device has a number of data storage elements, each of which might be loaded on or from the bus, the

local interface contains the necessary multiplexing logic and/or a program register to help in the matte matters.

2.2.1 CRT dot matrix display

A brief introduction of the CRT display must proceed any explanation about the actual interfacing.

The CRT display developed and fabricated by Sastri [11] is a modified dot matrix unit. The full display screen can accomodate 16 lines of text in an Indian script which consists of composite characters (C.C.) juxtapositioned to form words and sentences C.C.'s themselves may contain a basic symbol (consonant or a vowel) and/or a vowel modifier or diacritical mark and/or upto two half characters.

Half characters carry special signficance in Indian scripts. A careful study of all Indian scripts reveals the fact that most of the half characters can be obtained merely by slicing off some portion of the full character vertically or horizontally. This fact has been fully exploited in the display design and its symbol memory contains dot patterns of the basic symbols, vowel modifiers, diacritical marks and only those half characters which cannot be obtained from the full character by truncation. A height and width information is used to obtain the desired slices.

Each line on the CRT screen is conceptually partitioned into a number of subframes, each carrying a composite character. The process of displaying the C.C. on the screen involves the following steps:

(i) Access the first symbol in C.C. from the appropriate location in the symbol ROM.

(ii) Place the symbol in the subframe such that it touches the 0-th row and 0-th column.

(iii) Shift the symbol suitably in horizontal and vertical direction and store the subframe.

(iv) Repeat the process for remaining symbols in the C.C. and superimpose all of them to obtain the C.C.

The following information about each C.C. is therefore required by the display:

(i) Starting address for the dot patterns of each of its constituent symbols in the symbol memory.

(ii) Vertical displacement (Y-shift) for all its symbols.

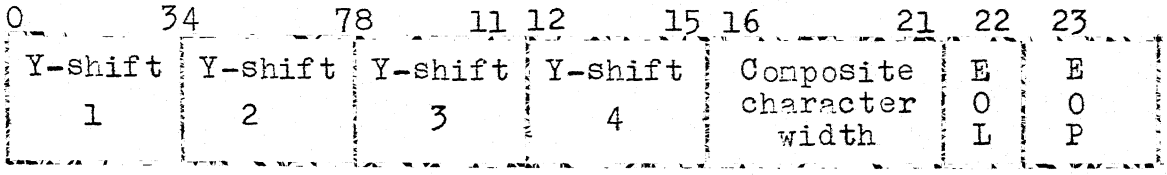
(iii) Horizontal disp. (X-shift) for each of its symbols.

(iv) A height and width information for each symbol.

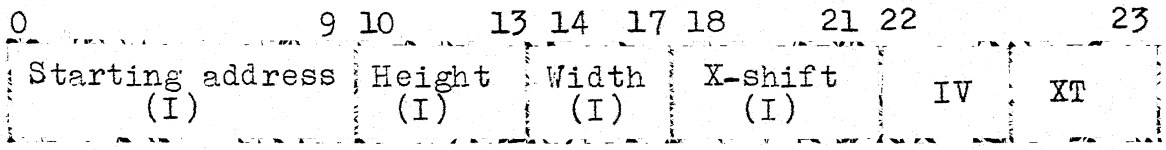
In addition two flag bits End of Line (EOL) and End of Page (EOP) become necessary for a complete text display.

Memory Organisation:

All this information about the C.C.'s constituting the text is made available to the display in its 4K x 24 RAM by the composition processor in the following format.



Location n, n = 0, 5, 10 ... of display memory



Location n+I of display memory, I = 1, 2, 3, 4
(Information about symbol I of C.C.)

Fig. 2.5: Word formats in display memory

Each C.C. is seen to occupy five words in the memory. If the C.C. contains less than four symbols, the unused words of the C.C. in memory are filled with zeros.

The IV (Inverse Video) bit serves to display the symbol in the inverse video font to mark it as a cursor character on the screen.

2.2.2 CRT Display Interface

Referring to Fig. 2.4 for the general schemetic for device interfacing to bus, since IFR unit has already been discussed we can straightway jump to the local interface. The Local Interface: The basic purpose of the display local interface is to allow data coming over the bus to be loaded

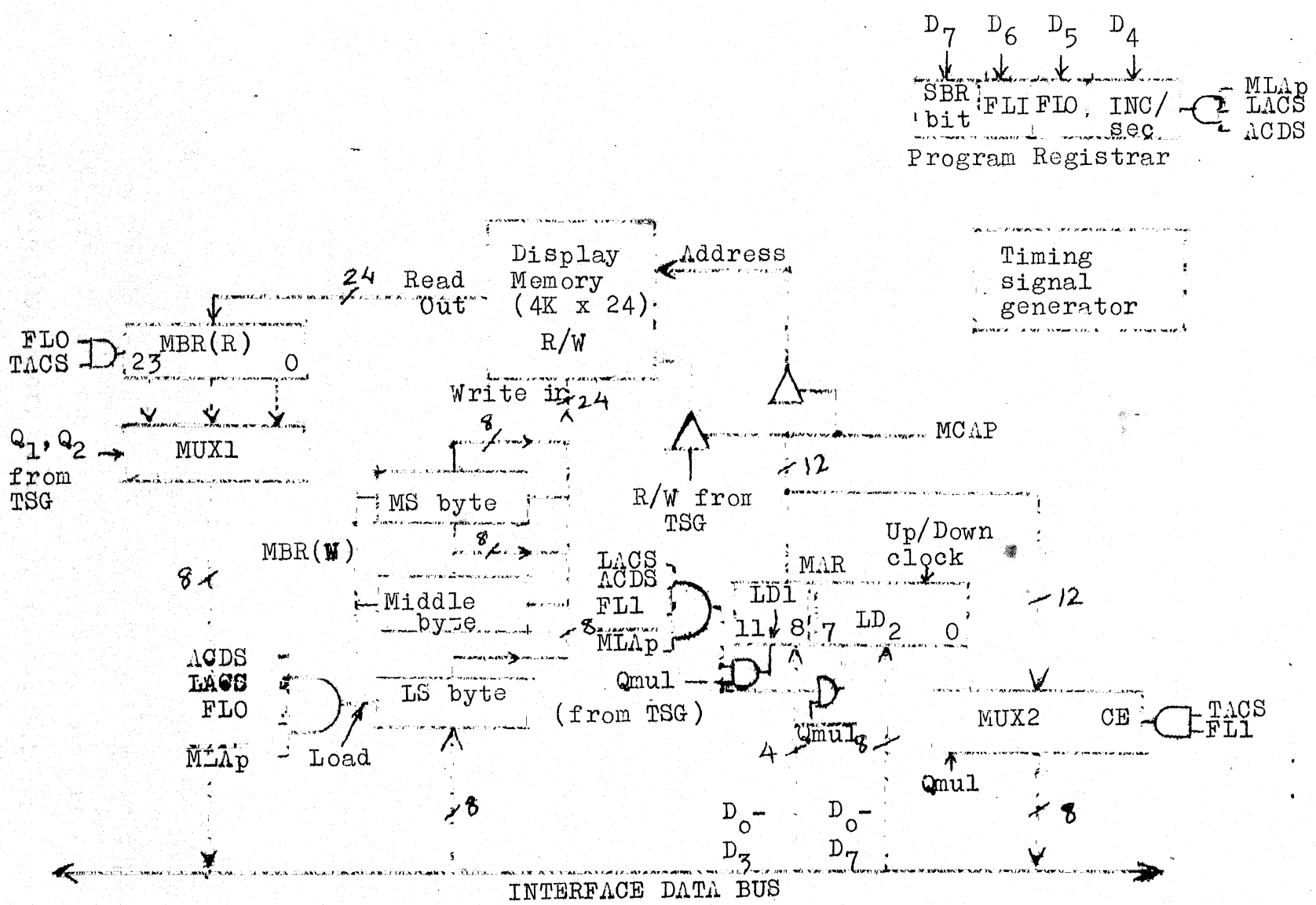


Fig. 2.6: Local interface logic for CRT display

into the display memory in appropriate locations and also, whenever necessary, to read data out of it to the system controller. Additional desirable interface features include a concordant multiplexing of the display memory between the display interface and the internal device functions of the display.

The simplified block diagram of the display local interface is shown in Fig. 2.6.

The interface is worked around four registers described below:

(i) Memory Address Register (12): MAR holds at any instant the address of the location in the memory where the next word coming from the bus will be entered, or, in a similar vein which word in memory will next be loaded on to bus while reading. The MAR automatically gets incremented (or descremented) after every interface read or write cycle of memory.

MAR, itself, is accessible to bus for reading or writing in two bus cycles - 4 MS bits in first cycle and 8 LS bits in the next.

(ii) Memory Write Buffer Register (24): MBR(W) is used to collect the data as it comes from the bus in 8 bit slices (MS byte first) and present it as the 24 bit word to the display memory during the subsequent write cycle.

(iii) Memory Read Buffer Register (24): is mainly used to latch the word read from memory during a read cycle. A multiplexer slices it and presents it to the bus in three 8 bit slices (MS byte first).

(iv) Programming Register: The PROG REG is very vital to the functioning of the display interface and serves to program it into performing anyone of its several functions. It is a 4 bit register and is accessible to the bus only for writing. The four bits carry the following operational significance:

(a) Flag 0 (FL0): When true configures the display interface to load data coming from bus into MBR(W). A write pulse for display memory is automatically generated after every bus write cycles and the MBR(W) word gets loaded into the address currently held by MAR. MAR gets incremented by one after the write pulse.

(b) Flag 1 (FL1): FL1, when true, programs the display to make MAR to be accessible to the bus for reading or writing.

(c) Inc/dec flag: The MAR is incremented or decremented by one after every complete memory read or write cycle according as whether this flag is Hi or Lo respectively.

(d) SBR flag (Single byte memory read cycle): During the execution of edit routines a situation often encountered is that the display memory contents must be rapidly scanned, forward or backward, for the bits EOL or EOP. Since both of

these bits occure in the MS byte of the memory word it would be a waste of time to read all three bytes of the word. This flag bit when true configures the display logic to increment/decrement the MAR after EVERY memory read cycle, so that only MS byte is sent over the interface each time.

To write into the PROG REG system controller preceeds the program byte by a MLA (program) address of the display. A master controller in the IFR unit accepts this address and generates a MLAp signal during the ensuing display write cycle to route the byte to PROG REG. Once programmed the contents of PROG REG remain unaltered until the next PROG REG writing.

The generate operation of the interface is easy to visualize. In the receiving mode (LACS true) the data bytes from bus are received only when ACDS is true and are loaded into appropriate interface registers depending on the status of PROG REG and MLAp. During send mode (TACS true) data from desired registers is loaded on the bus one byte at a time. After one byte is successfully transmitted (SWNS true), the next one is loaded through the use of multiplexers.

2.2.2.1 The timing signal generator

Logic circuits within the TSG generate the following timing signals:

(i) Memory capture signal: A single bit PROC INT signal

is sent to the display device functions, whenever the display memory must be accessed by the interface for reading or writing, according to the logic $(LACS \wedge FLO \wedge MLA\bar{p}) \vee (TACS \wedge FLO)$. The display reacts by sending a Device Halt and releasing the address and R/W lines of the display memory to the bus. This DVC HALT signal is used by the interface to capture the memory as shown in Fig. 2.7.

(ii) INC/DEC logic for MAR: A modulo-3 counter forms the heart of the INC/DEC logic given in Fig. 2.8. When SBR is true Mod-3 counter is held to $Q_1Q_2 = \emptyset\emptyset$ and this configures the MUX1 to always load MS byte of MBR(R) on the bus. The reset pulse generated after every 3 bus cycles involving memory transactions (MCAP true) is used to generate the write pulse.

(iii) Generation of rdy and nba: nba and rdy is generated according to the state diagrams described in Fig. 2.9.

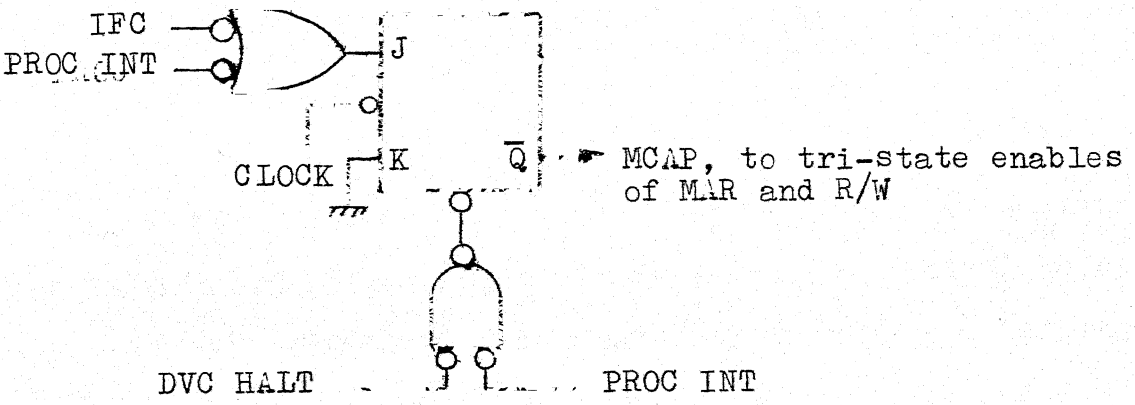


Fig. 2.7: Memory capture logic

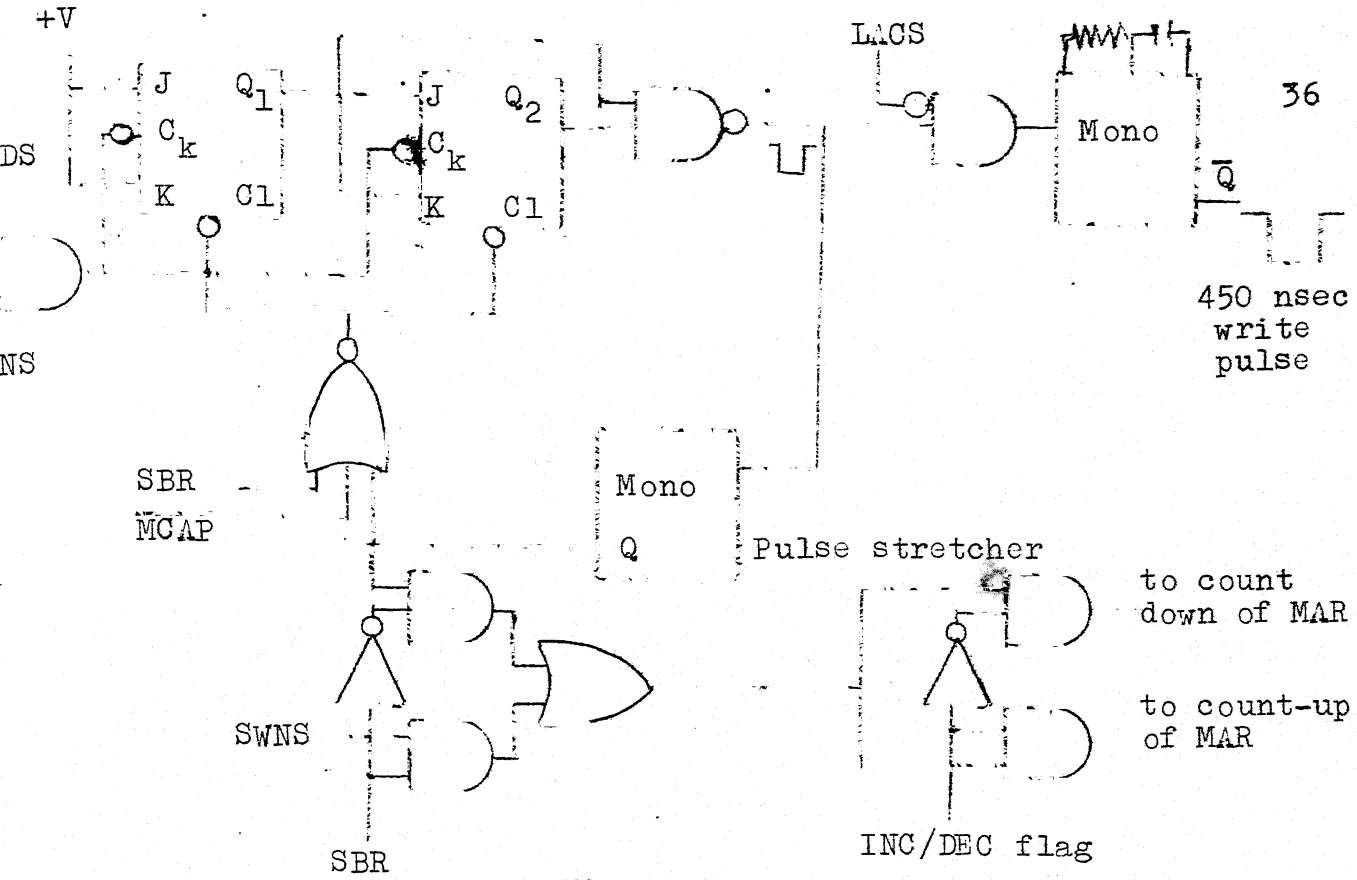


Fig. 2.8: Increment/decrement logic

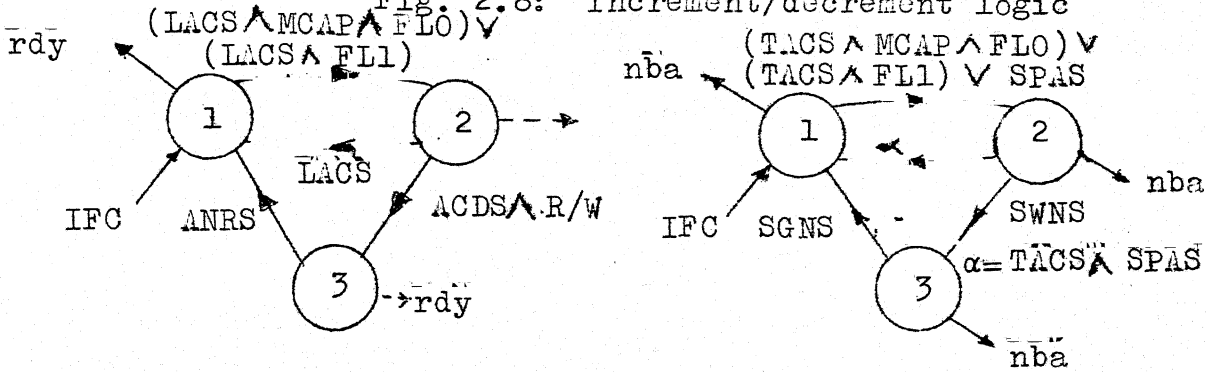


Fig. 2.9: Generation of local messages - nba and rdy

In words: In case of rdy, rdy must be sent true when LACS is true and, if display memory is to be involved in transactions, additionally it must be duly captured before the rdy is sent (MCAP true). rdy is removed with ACDS and the cycle can repeat after ANRS. If a write pulse is being generated in a particular cycle, the next cycle must be delayed

by as much. This feature is incorporated in 2 → 3 transition. A similar statement can be made about the generation of nba.

2.2.2.2 Use of Master sequencer to generate MLAp signal

The IEEE std. 488 specifies that a 5 bit talk or listen address be given to a device in the following format:

MLA = XØ1LLLLL

MTA = X1ØTTTTT

Accordingly following addresses were given to different devices of our system.

	MLA	MTA
Key board	XØ1ØØØØ1	X1ØØØØØ1
Composition Proc.	XØ1ØØ1ØØ	X1ØØØ1ØØ
Display	XØ1ØØØ1Ø	X1ØØØØ1Ø

An additional MLAprog address = XØ11ØØØ1 was given to the display, which was to be sent only when its program Register was to be filled. A master sequencer is used to sense this MLAprog and generate the required MLAp signal during the actual transaction. The following state diagram [Fig. 2.10] describes a master sequencer:

The reception of MLAprog from system controller sends the master sequencer to PAES state. The system controller now removes ATN and loads the program byte on the bus, on receiving which the master sequencer transits to PACS. The

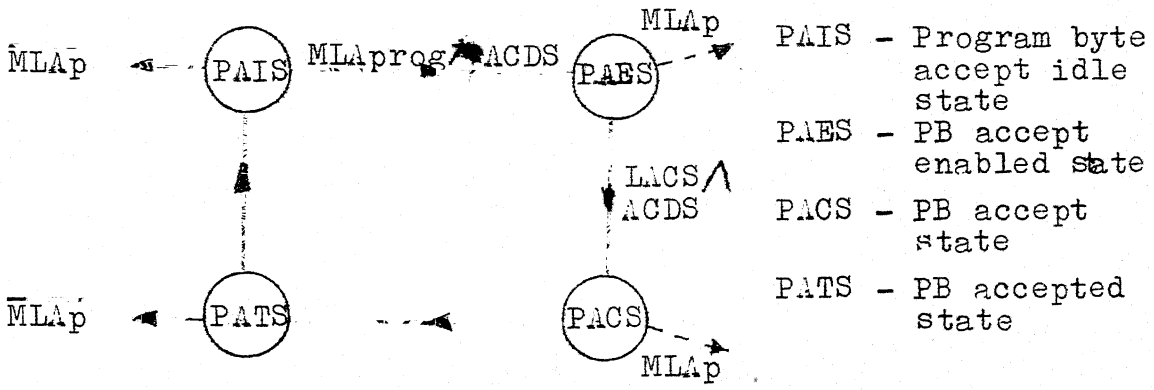


Fig. 2.10: Master sequencer state diagram

cycle of MLAp generation is thus completed.

The modular approach (discussed earlier) is used to implement the state diagram and generate MLAp.

2.3 The Composition Processor Interfacing

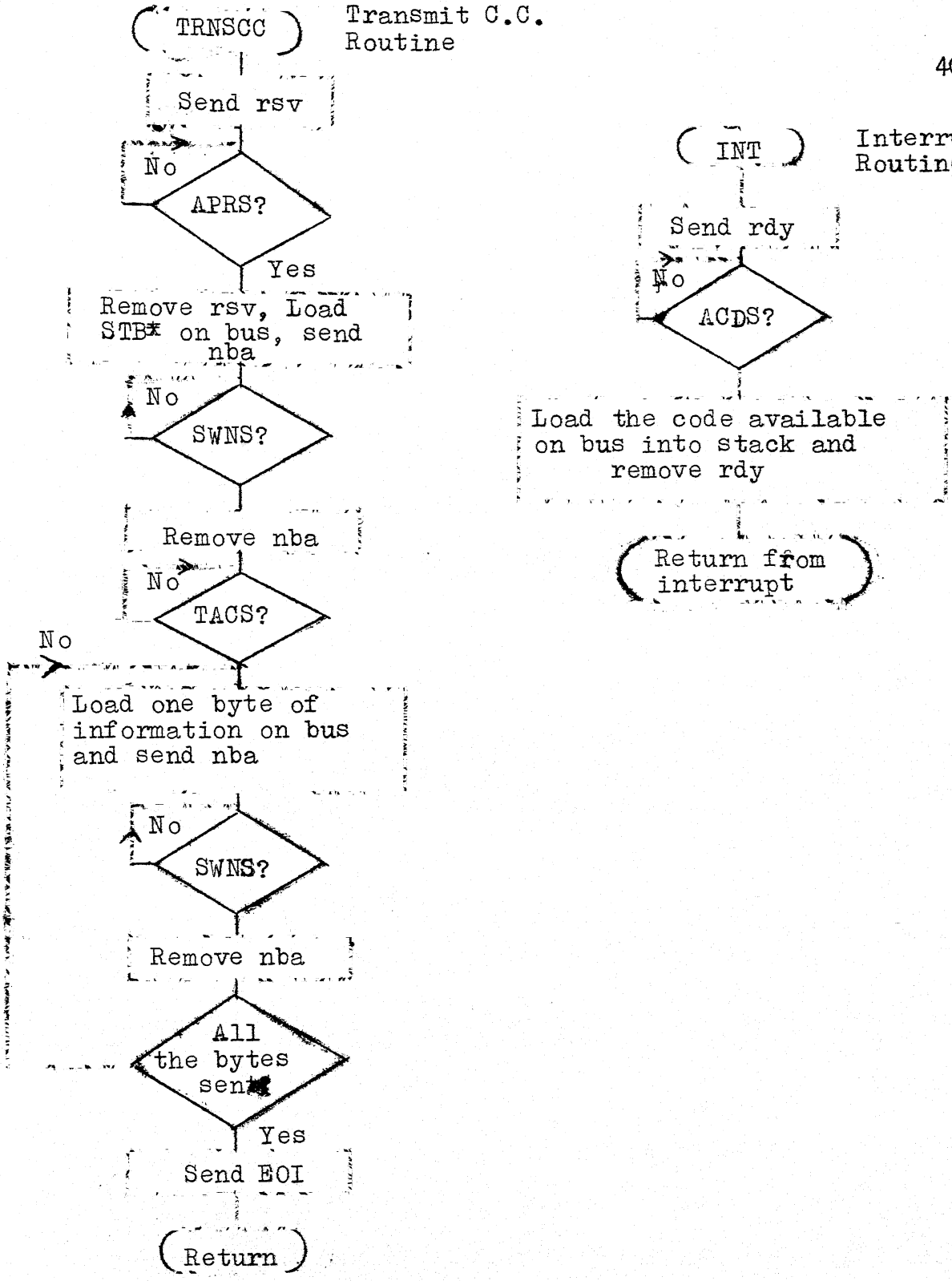
The algorithms for 'Composing' the input character have been developed by Laturkar for Devnagiri Script [12]. The programs have been written in FORTRAN for IBM 1800 and an IBM 1627 plotter has been used to output the characters for demonstration purposes. A microprocessor based composition processor is soon expected to some up for our stand alone I/O system which will implement the above algorithms to generate the display compatible information we need. It is for this module that we intend to suggest a possible bus interfacing scheme - which, of course, until the module comes up in reality, can at best be called tentative.

While the schematic of Fig. 2.4 is still suggested, the local interface part can be simulated in software. As such special dedicated routines inside composition processor will interact with the IFR unit to generate local messages like rdy, nba and rsv to effect the necessary bus transactions, plus, if necessary the slicing of display bound information into three 8 bit bytes (to be sent with MS byte first).

The general flow of events for the composition processor shall proceed as follows:

An LACS true signal from the IFR unit generates an interrupt for the comp. processor which branches to the interrupt routine to receive the character code from the keyboard and pushes it into a stack. The comp. processor keeps checking this stack and when all symbols of a full C.C. have been received it executes appropriate routines to generate the display compatible information about the C.C. and then branches to a TRANSMIT C.C. routine to send the information to the display.

The interfaces for KB and system controller are described along with the respective device descriptions in Chapters 3 and 4 respectively.



* STB is merely RSQ bit true (lo) on D_6.

Fig. 2.11: Bus interaction flow charts for composition processor

CHAPTER III

THE KEYBOARD

A keyboard, in view of the vast variety of applications to which it can be put, can be broadly described as a means to encode a given message string into a machine readable form for the purpose of processing, storage, display, transmission etc. Keyboarding, then is the process of encoding the message string into such form.

3.1 Available Schemes for Keyboarding: A Survey

The process of mechanization of Indian scripts is at present passing a development phase - on the lines of the classical phenomenon of 'try and reject'. A number of keyboarding schemes are being explored, refined and later thrown into the arena by their promoters for the struggle of the fittest. Most of the schemes, understandably, derive heavily from the exisitng technology on the mechanization of Roman script. While this is no mean thing to do, they tend to tackle difficulties which are inherent in Indian scripts [3] by brute force, these difficulties not being present in Roman script. As a result the keyboarding procedures in them are very cumbersome and inefficient because of the large number of keys required and an unnatural scheme of depressing them for unputting a compsite character. The schemes are however, being refined by their promoters to remove the various defects

in them and two of them deserve a fair mention:

(i) Using a Roman keyboard as such for inputting the Indian script after romanizing it.

(ii) Relable keytops of conventional Roman keyboard with symbols of Indian script and use it for keyboarding.

A markedly different approach to mechanization of Indian scripts was proposed by Narsimham et.al. [4] and Sinha et.al. [5]. They pointed out the phonetic based structure of all Indian scripts and proposed a phonetic based keyboarding scheme which was universely applicable to all major Indian scripts. The keyboard in their scheme, designed specifically for Indian scripts, contains a keyset comprising of all consonants, vowels and diacritical marks of the script, with no keys provided for the half characters and vowel modifiers. An action key is used in both the schemes, though differently, to mark the formation of composite characters.

In Narsimham's scheme a '*' operator (a connotation for depressing action key) before the symbol is used to indicate that the succeeding symbol must be combined with the preceeding one to form a composite character. Thus सिद्धार्थ in this scheme gets encoded as स * इ द् * ध आ र * थ .

In Sinha's scheme the '*' operator before a symbol is used to mark it as a half character or vowel modifier. सिद्धार्थ in this scheme is thus encoded as स * इ * द् ध * आ * र थ.

Obviously a greater understanding of script orthography is needed in this scheme. However this scheme always gives a unique coding to the text. The presence of a processor to accept the encoded symbol string and do further processing on it is assumed in both the schemes.

The relative advantages of the two schemes are rather obvious. Besides being a scientific approach to the problem of mechanization, they are expected to give higher keyboarding efficiency because of the reduced set of keys confronting the operator, and a simple phonetic scheme of inputting especially in Narsimham's scheme. The schemes are universally applicable to all the Bramhi based Indian scripts and hence the same keyboard with different key top lables can be used for different languages.

The design of a universal keyboard for all Indian scripts based on the Narsimham's scheme of inputting is presented in the following section. (This KB, with no changes whatsoever, can however be used for Sinha's scheme as well).

3.2 The Universal Keyboard for Indian Languages

The universal keyboard for Indian languages to be described here, contains a keyset consisting of 60 keys (52 character keys and 8 control keys) arranged in four rows as shown in Fig. 3.1.

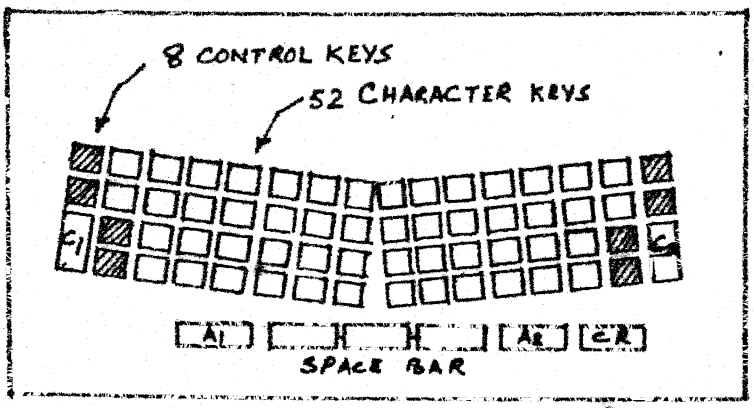


Fig. 3.1: The universal keyboard for Indian scripts

The entire key-set is devided into two equal right and left sections arranged to be at a mutual inclination of 10 degrees to provide for a natural hand setting to the operator. Additional keys account for the two action keys, two shift keys (for upper case and lower case characters), one space bar and one carriage return key. The eight control keys devid between themselves the following console commands: carriage return, End of page, cursor move right, cursor move left, cursor move up, cursor move down and GO in lower case and Display clear, character insert/delete and line delete in upper case. The unused control keys are reserved for future applications. The allocation for character key has not been attempted and will depend on the results of a statistical analysis for each language about the most frequently occuring characters in the script, so that the more convenient key positions will go to them. It is however suggested that all the basic symbols of the script and some of the more frequent punctuation symbols like the space, comma, full stop etc. be

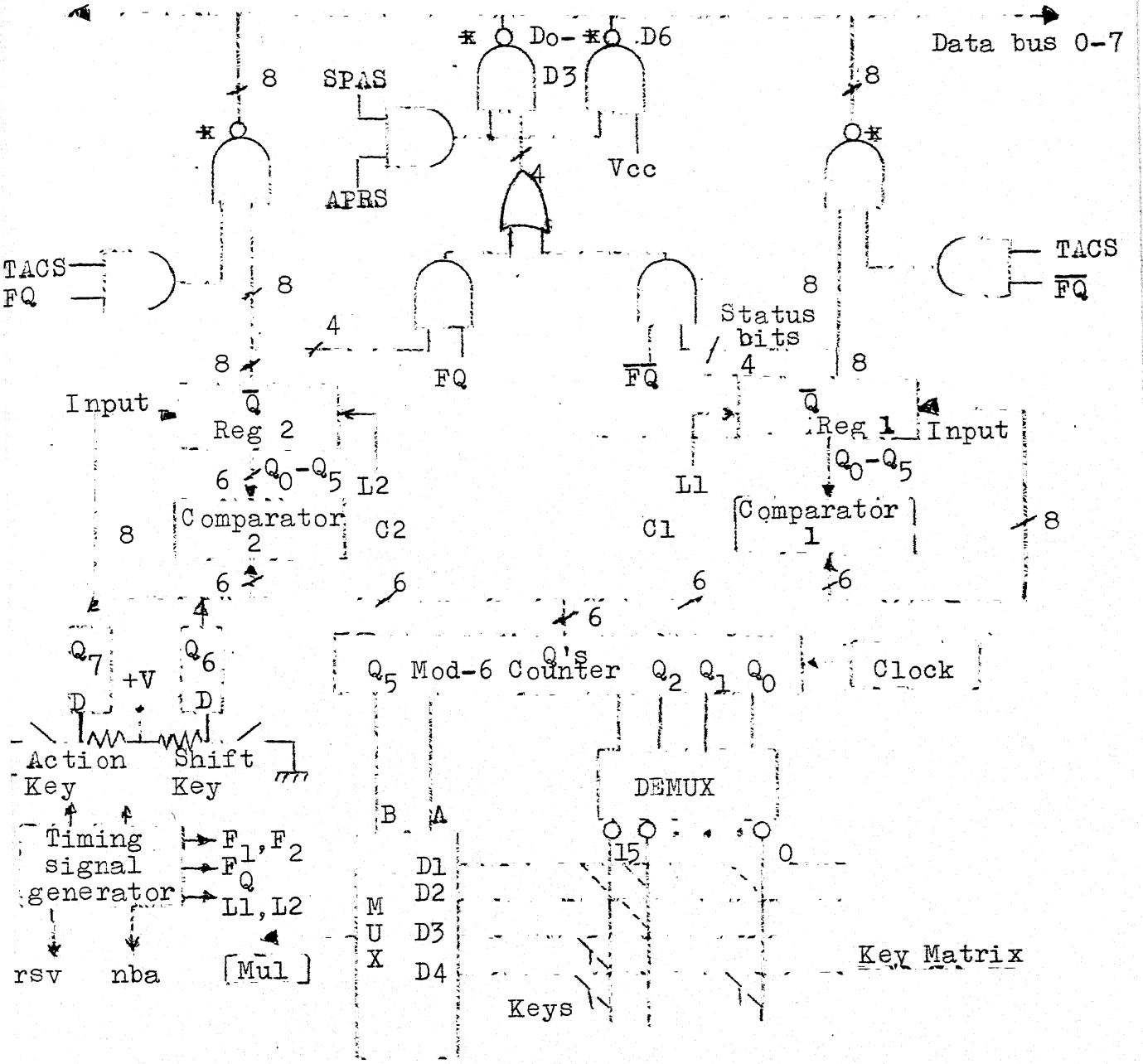
accomoded in lower case. The upper case will then contain numerals, more punctuation marks etc.

3.2.2 Key encoding: An 8 bit code is generated for each key depression by the keyboard logic. Bits 0-5 of this code correspond to the position of the key in the matrix of 60 keys. The sixth and seventh bits carry the status of action and shift keys respectively. The codes for control keys have been allotted in such a way that they have '0' in bit positions 0, 1, 2 and 7 to facilitate their recognition by the system controller, as explained in the first chapter. The code allocation for other keys have been arbitrary.

3.3 Keyboard Electronics

A schematic for the keyboard electronics is shown in Fig. 3.2. A facility for two key roll over is provided in the basic logic design to increase the keyboarding speed. In this arrangement, while a key is depressed, if a second key is depressed too, the code for the second key must also be transmitted in sequence . The assumption is that by this time the code for the first key has been successfully transmitted.

The heart of the circuit is the key matrix and the associated key scanning logic consisting of a 6 bit counter, a 4 bit demultiplexer and 2 bit multiplexer. The 60 keys arranged in a 4 x 16 matrix as shown are being continuously



$$L2 = F_1\bar{F}_2\bar{C}_1 \cdot Mul \qquad F_2 = C_2Mul + L2$$

$$L1 = (\bar{F}_1F_2\bar{C}_2 + \bar{F}_1\bar{F}_2)\cdot Mul \qquad \bar{F}_2 = C_2\bar{Mul}\cdot\bar{L}2$$

$$F_1 = C_1Mul + L1 \qquad F_Q = L_1\bar{L}_2$$

$$\bar{F}_1 = C_1\bar{L}1\cdot\bar{Mul} \qquad \bar{F}_Q = \bar{L}_1L_2$$

Fig. 3.2: Keyboard Logic Schematic

scanned by the Mod-6 counter. A 'lo' is presented to each of 16 columns once in every 16 clock pulses and if a key is depressed, the 'lo' is transmitted to the corresponding row and in turn appears as a 'Mul' signal once in every 64 clock pulses. Hence the appearance of a 'Mul' signal (which remains 'lo' for a complete clock period) is an indication of a depressed key having been encountered and the count of the mod-6 counter during that period gives the code for the key depressed. This code when combined with the status of the action and shift keys yields the complete 8 bit code for the key which is loaded into one of the two registers Reg 1 or Reg 2 for the next task of sending the code to the composition processor.

The scanning rate determines how fast the depressed key is detected. To keep a reasonably fast rate, without complicating the circuitry more than what it is, a clock rate of 1 MHz was selected. Since the key will be kept depressed for a much longer duration than a single scan period of 64μ sec, a number of Mul pulses will be generated for each key depressed one in each scan cycle. The code however must be sent only once corresponding to the first 'Mul' pulse and the subsequent ones must be ignored. Also if during this period a second key is also simultaneously depressed the first key may be 'forgotten' and the code for the second key is sent but here too only once.

All this is acheived with the help of key - status flip flops F_1 and F_2 and the two comparators. F_1 remains set as long as the first key remains depressed and F_2 is set when a second key is depressed (while the first is still depressed). Whenever a 'Mul' signal occures, the logic checks the status of flip flops F_1 and F_2 and takes the following steps:

(a) If both F_1 and F_2 are reset, it is a new key encountered. A load pulse L_1 is generated and the key code is loaded into the Reg 1. F_1 is also set.

(b) If F_1 is set and F_2 is not, it checks the comparator output C_1. If it is true, it is the same key encountered during a subsequent scan. If C_1 is false, it is a 'second' key just depressed. The load pulse L_2 is now generated and the code is loaded into Reg 2. F_2 is now set.

(c) If F_2 is set and F_1 is not. It is the situation when the 'second' key is still depressed and another key has been struck. This key is now treated as the first key and accordingly L_1 is generated and F_1 is set.

(d) If both F_1 and F_2 are set no action is taken.

The status of $F_1(F_2)$ while it is set is refreshed at every scan of the key matrix. The moment the key is released (A key release is indicated by the absence of 'Mul' signal when $C_1(C_2)$ comes), $F_1(F_2)$ is reset. A flip flop F_Q keeps track of the most recently filled register and determines which register will be loaded on the system bus when instructed

to do so by the system controller.

The remaining logic replaces the local interface of Fig. 2.2 and is best discussed along with keyboard interfacing.

3.4 Keyboard Interfacing

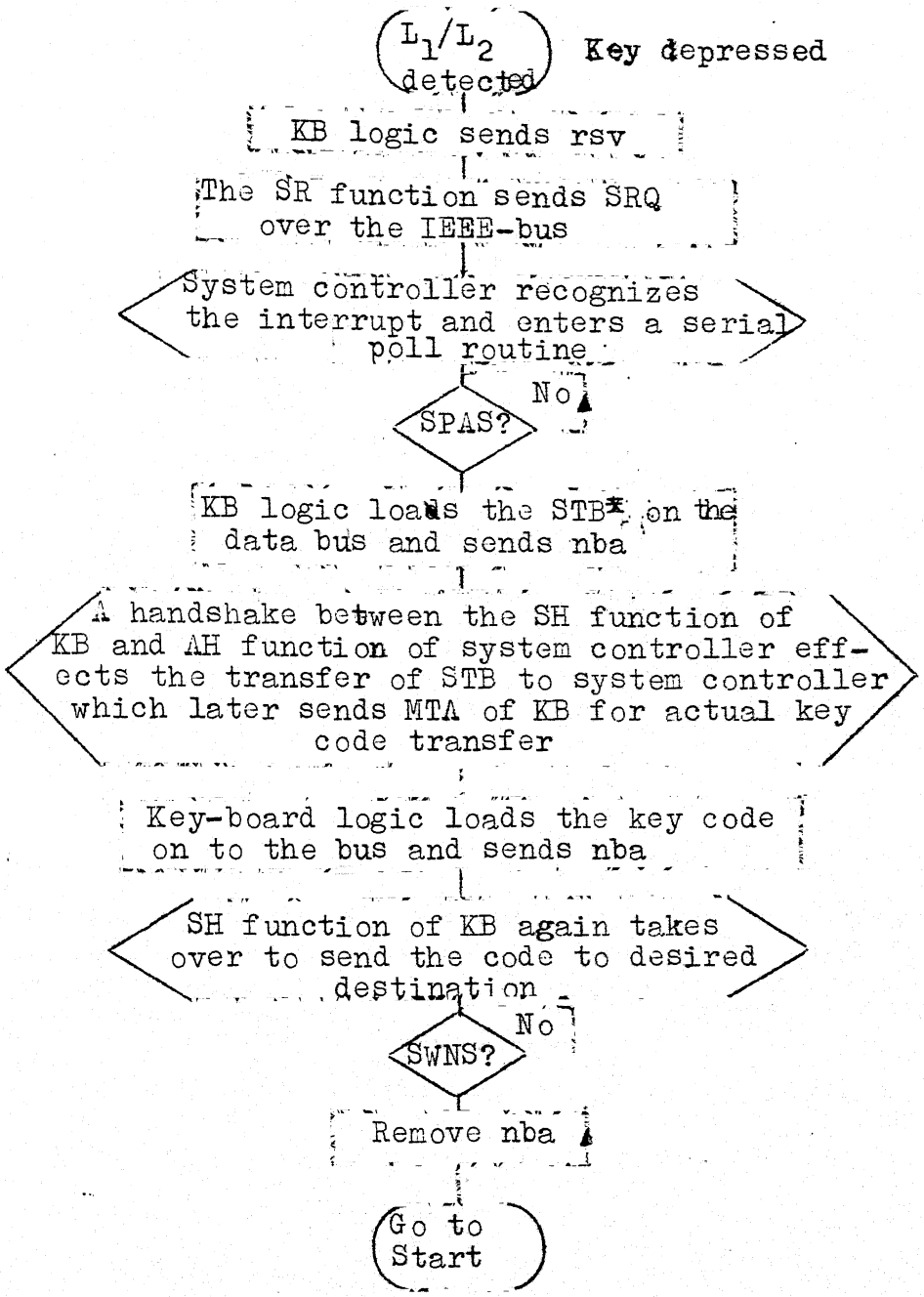
While the general keyboard interface still takes the form given in Fig. 2.2 the local interface is conspicuous by its absence. This of course can be seen to form the integral part of the keyboard electronics. This part essentially does the following things:

(i) Generates the signal rsv - when a new key is depressed ($rsv = L_1 + L_2$) and nba when the STB or the key code must be sent.

(ii) When the serial poll reaches the KB (SPAS & APRS) the logic leads the STB on to the bus.

(iii) Subsequently when TACS becomes true the appropriate output register is loaded on the bus (determined by F_0).

When a new key is depressed L_1 or L_2 is generated. The KB interface logic takes over from this point and the general flow of events to follow are described by the flow chart of Fig. 2.3.



* The STB takes the form

$D_1 \ldots\ldots D_0$

S1SSSSSS

where Ss are the status information bits. In our case positions D_0-D_3 of STB carry the bits. 0,2,3 and 7 of the key code being transmitted. (For a control code all these bits are logic Lo)

Fig. 3.3: Flow of events in the interface system following a key depression

I.I.T. KANPUR
CENTRAL LIBRARY
Acc. No. 55456

CHAPTER IV

SYSTEM CONTROLLER

The system controller forms the heart of the I/O system and contains routines for programming the interface functions, system initialization, bus scheduling and text editing. A motorola 6800 micro-computer has been used for implementing these routines. Flow charts for these routines as well as for the main program are described in section 4.2 of this chapter. Interfacing of the microcomputer to the IEEE-bus is described in 4.1.1 and generation of bus compatible signals through system software, for effecting bus transactions involving system controller is explained in Sec. 4.1.2.

4.1 Interfacing System Controller to the Bus

The microcomputer makes use of Peripheral Interface Adapters (PIAs) [14] for talking to the outside world. Use has been made of two such PIAs to connect the M6800 directly to the IEEE-488 system bus in the following manner [Fig 4.1]

The output lines of both the PIAs are buffered through bidirectional tristate buffers to the system bus. Since these lines directly go to the IEEE-bus lines, they are programmed to be in the input mode in all bus transactions not involving the system controller to avoid conflicting outputs on the same line. The SRQ line is connected to the programmable interrupt input line CB_2 of PIA(B). A 'lo' on SRQ line

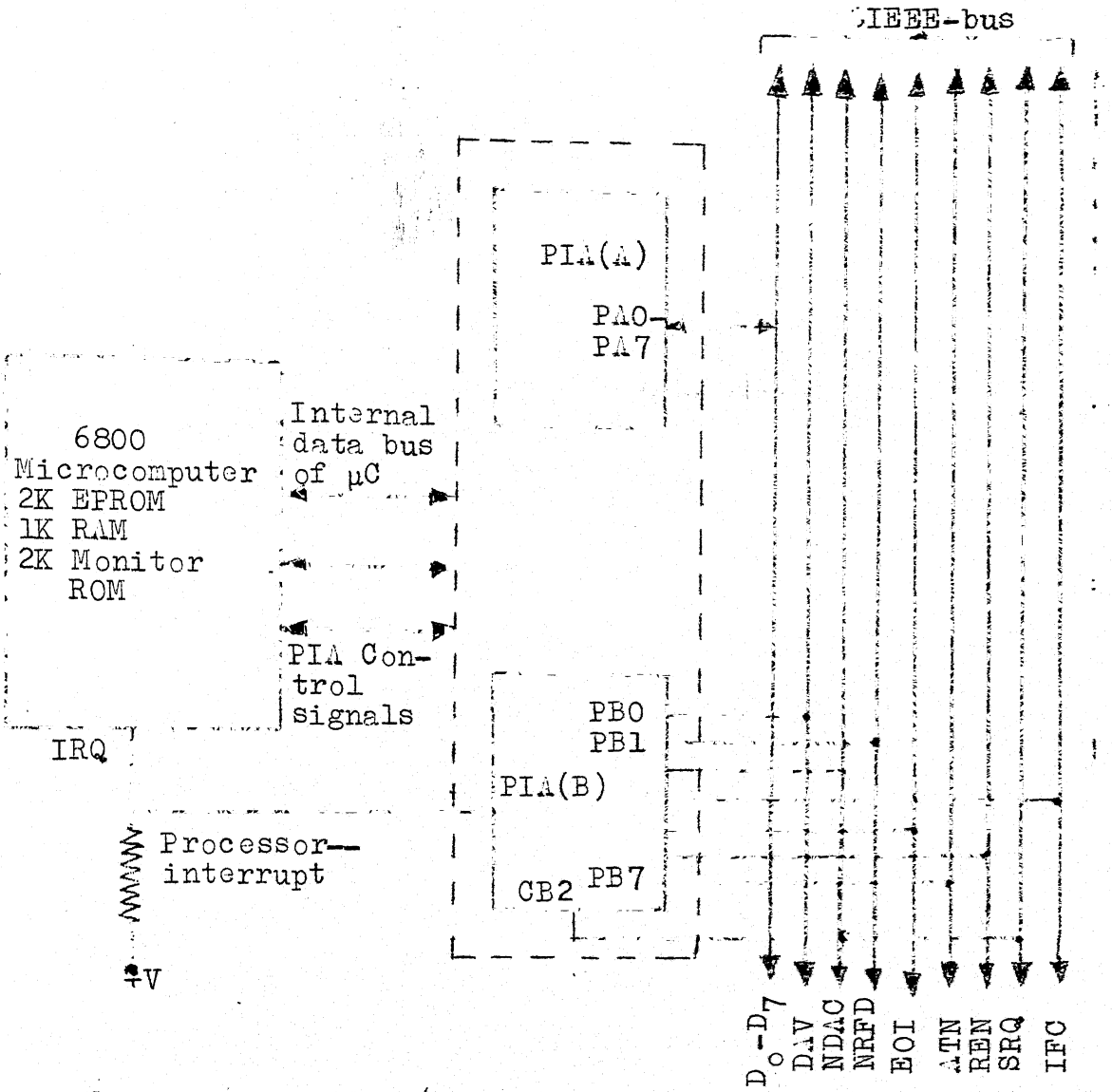


Fig. 4.1: IEEE-bus/system controller interface

at any instant causes an interrupt to be sent to the processor at its IRQ port, which acknowledges it if its internal interrupt mask bit is reset or else the interupt is kept waiting .

4.1.2. Generation of IEEE standard compatible interface signals through software:

The IEEE Std 488 specifies that all transactions of a device with the interface bus must be routed via the

appropriate interface functions, and, for that matter, a protocol be followed strictly in accordance with the corresponding state diagrams for all transactions what-so-ever. A careful study of the standard however reveals that behind all these stringent specifications there is the all important need to generate (and receive) interface bus signals like DAV, NRFD, NDAC, ATN etc. and the local messages like LACS, TACS etc. at just the appropriate instants (specified by the state diagrams). The means adopted to generate these signals are unimportant. While it is strongly recommended that in all devices the requisite interface functions be explicitly implemented through hardware to generate these signals for simplifying the interface design, some variations can be tried for implementing the state diagrams for the system controller.

A scheme for generating these signals through system software was tried for the system controller. In this scheme whenever the system controller has to receive a byte from the bus or send over it to other devices, it executes corresponding routines of RBYTE or SBYTE, the flow charts of which are given in Fig. 4.2.

As various signals are being generated/read by the two routines, the state of corresponding interface functioning are only implicitly being entered (shown against different blocks in the flow charts). Many states are being bypassed

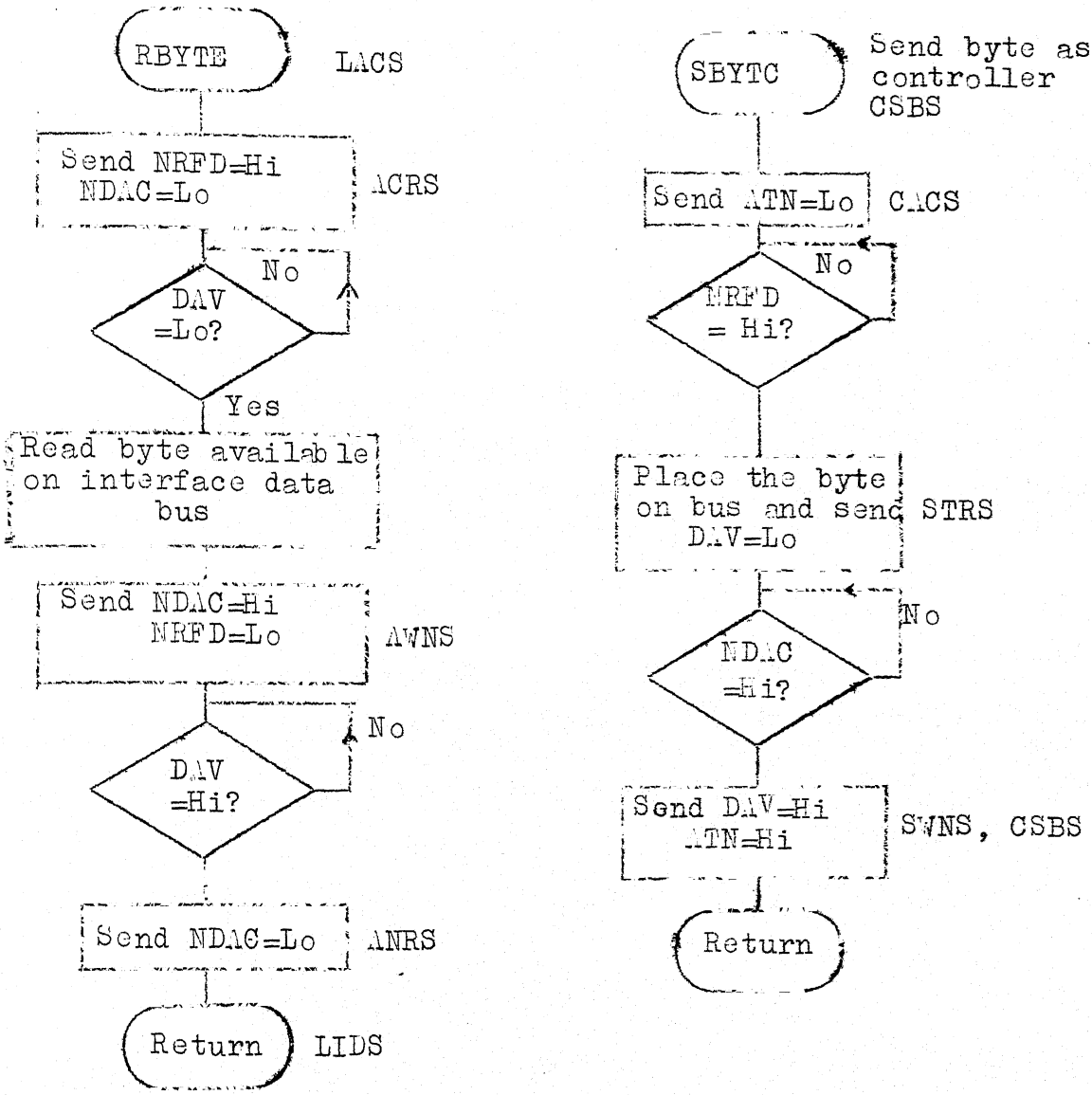


Fig. 4.2: Routines for generating handshake signals

(like the power on idle states etc) without in anyway offending the state diagrams. Care is however taken in the main system program to ensure the presence of 'idle state' interface signal levels at the bus before any of these routines are called.

These two routines and a third one for sending byte as device (SBYTD) are very basic to the operation of the system controller and will be encountered very frequently in the remaining chapter. (SBYTD is similar to SBYTC except that status of ATN remains Hi throughout the program).

The scheme while doing away with the interface hardware necessary to implement the interface functions, can claim to be faster in some cases since the actual state transitions (in IFR unit) are being bypassed. However the approach needs to be applied with due caution. Since, once the routines are entered they are executed in full, a very many state transition paths in the corresponding interface functions are being ignored. This will create problem if the system were to entertain service request at arbitrary instants (say in the middle of a transaction).

4.2 System Controller Routines:

The different system controller routines are presented in this section. We start with the main program.

4.2.1 System flow chart (and bus scheduling): The system controller is first to wake up at the power on, and in turn tones up the rest of the system to make it ready for receiving the first key depression by the operator. The sequence of events from power on are illustrated in the following flow chart [Fig. 4.3].

As a principle one conversation is allowed to finish before the system is reconfigured to changed the data paths. This is desirable because of the large system overheads for establishing a new data path following a service request (The overheads involve the execution of serial poll routine, sending of MTA and MLAs etc.). The service requests are thus entertained only at place (X) on the system flow chart. In our system this means that the KB might have to wait while the composition processor is dumping a newly composed C.C. into the display memory or both KB and the Comp. processor have to wait while the system controller merrily goes about one of its Screen Management Routines. While the first wait time ($\simeq$ 400μ sec) is almost insignificant and will not effect the keyboarding speed, the second is not so, as many SMARTs tend to take almost a second or more to end. This however is not objectionable as these routines are called only during 'halts' in the keyboarding and no interrupts are infact waiting.

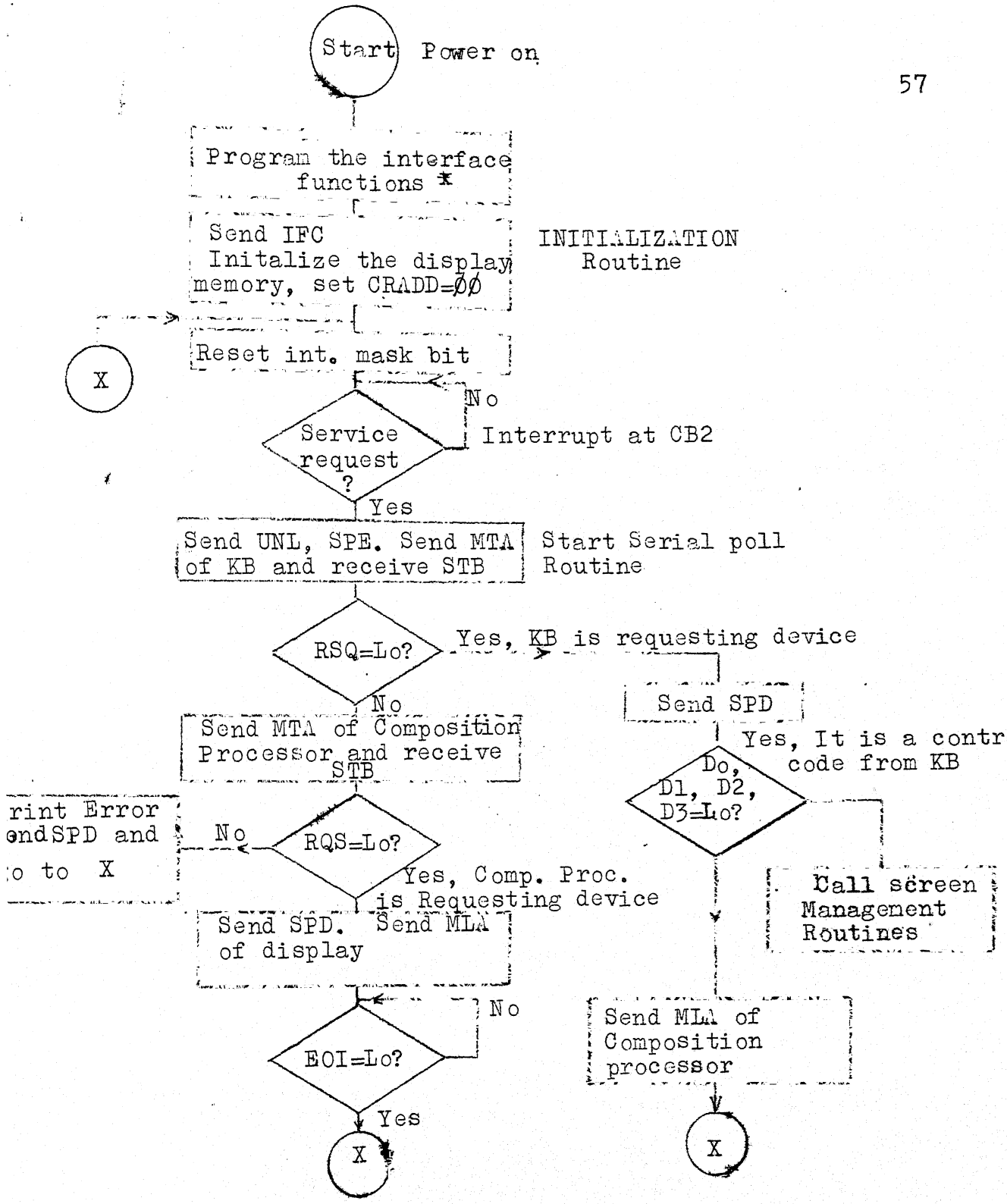
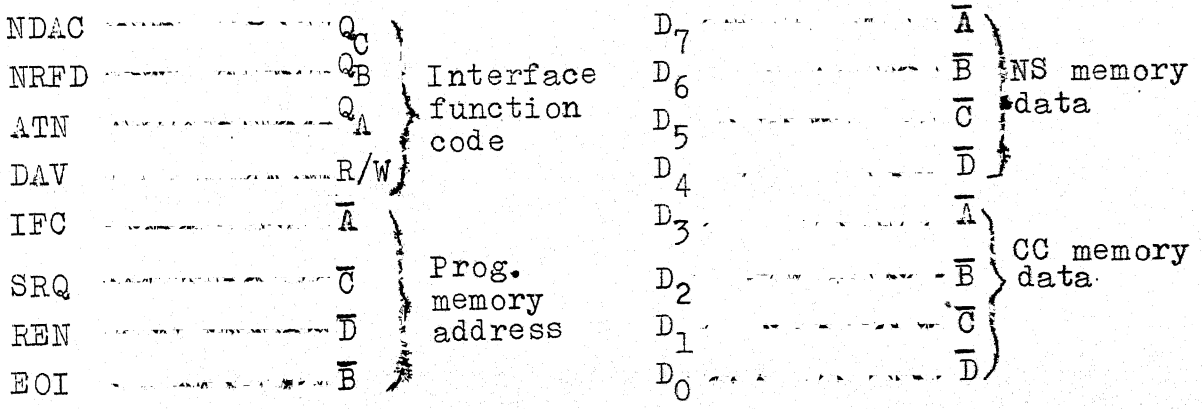


Fig. 4.3: System Flow Chart

The basic scheme of the system operation follows the general description of Section 1.4. A step by step procedure to set up the system after power on and the various error messages printed by the system controller on the teletype is given in the User's Manual (Appendix A).

4.2.2. Programming the interface functions:

The first step in setting up the system for operation is calling upon the system controller to program all the interface functions. All bus activities are terminated and the system controller uses the 16 lines of the bus to send data to NS and CC memory (4 bits each), address of memory locations where these data must be entered (4 bits), a three bit code for the interface function being programmed and the R/W pulse in the manner shown below:



The routine programs all interface functions in the system implemented by the modular approach one by one. (Similar interface functions in different devices get

programmed at the same time, however, because of the common interface bus being used for sending data and addresses). A decoder in each IFR unit decodes the code for interface function and enables the corresponding interface function for receiving the program bytes and address. All the data and addresses are loaded on the bus lines by the routine and the R/$\overline{W}$ line is then pulsed 'lo' to enter the data at given location in the two memories of the desired interface function. This is repeated for all the interface functions to be programmed. The listing of the program routine is given in Appendix C.

4.2.3 Initialization routine

The programming of interface functions is immediately followed by initialization routine which takes following steps to initialize the system.

(i) Sends IFC to place all the interface functions in their idle state. It also clears MAR so that it points to the first location of display RAM.

(ii) Loads EOL and EOP bits in all the display RAM locations. This step besides initializing the device functions of the display also maintains EOP in the very first location after the last character filled into the display RAM, as the RAM gradually gets filled from the composition processor. This is necessary to give the display beam the necessary impetus to fly back to the top left corner of the screen for the next scan.

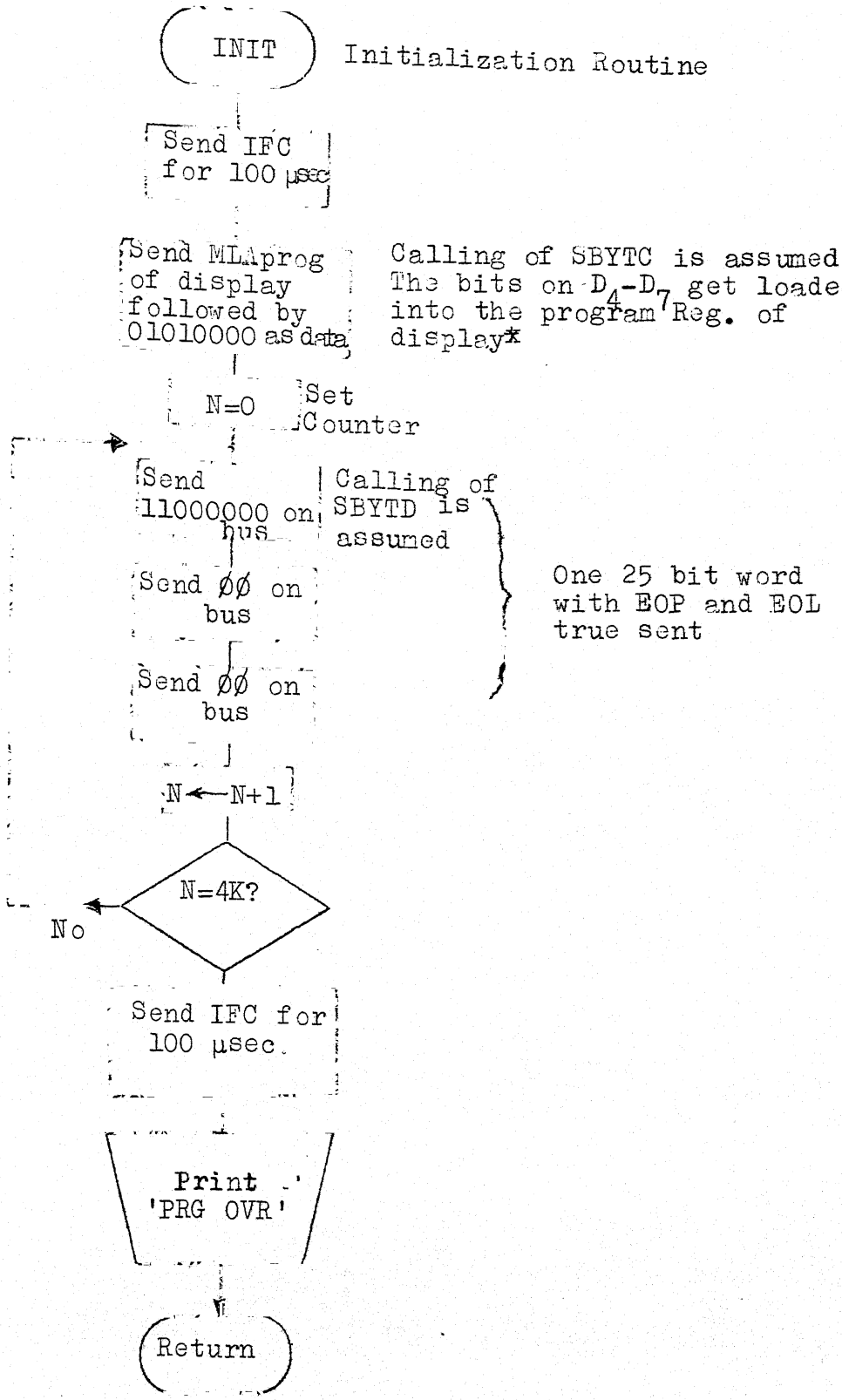


Fig. 4.4: Flow chart for initialization routine

* Allocation of bits given in Chapter 2 is repeated here for convenience D_4 - INC/DEC, D_5 - FLO, D_6 - FL1, D_7 - SBR.

(ii) Clears the register CRADD which amounts to bringing the cursor to the top left corner of the screen. This step will be explained later.

The flow chart is given in Fig. 4.4.

4.2.4 Screen Management Routines (SMART):

Package routines for screen management are called by the system controller at the instance of the operator when he depresses any of the following control keys: End of line (Carriage Return), End of page, Cursor move right/left/up/down, Display clear, Go, character insert, Character delete and line delete.

All these routines involve an elaborate scanning and manipulation of the display memory contents. Because of the variable number of composite characters in different lines of the text, some sort of scanning must be resorted to locate a given character on the screen in the display memory. The method is to hunt forward or backward for the End of line bit and from there on proceed backward or forward, character by character, until the given character is reached.

The cursor is carefully maintained all the time by the system controller as this is undispensable for all editing procedures. For an instance, to delete a composite character, the cursor is first brought to the position of the character to be deleted through the use of cursor move keys and then

the character delete key is pressed. The cursor character is just a character displayed in the inverse video font. The system controller maintains two registers CRADD and NWADD to help in the matters.

CRADD: This register contains at all times the address of first word of the cursor composite character (Each C.C. spans five words in display memory).

NWADD: This register is used as a temporary buffer to store the address of first empty location in the display memory while its MAR is being used for memory manipulations during various screen management routines.

A description of different SMARTs now follows:

4.2.4.1 End of line (Carriage Return):

This routine inserts an EOL bit in the last composite character filled into the display memory. The flow chart is given in Fig. 4.5.

4.2.4.2 End of Page

This routine inserts EOL and EOP bits in the last composite character entered in the display memory. The flow chart is similar to that for End of line except for that minor variation and has been omitted.

4.2.4.3. Cursor move right (->):

This routine moves the cursor from its present position to the next composite character to its right. Following steps

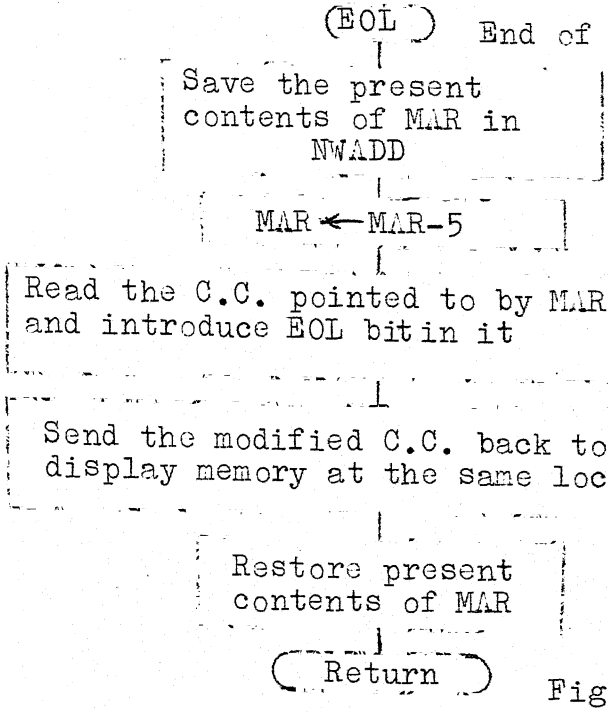


Fig. 4.5: Routine for EOL/EOP

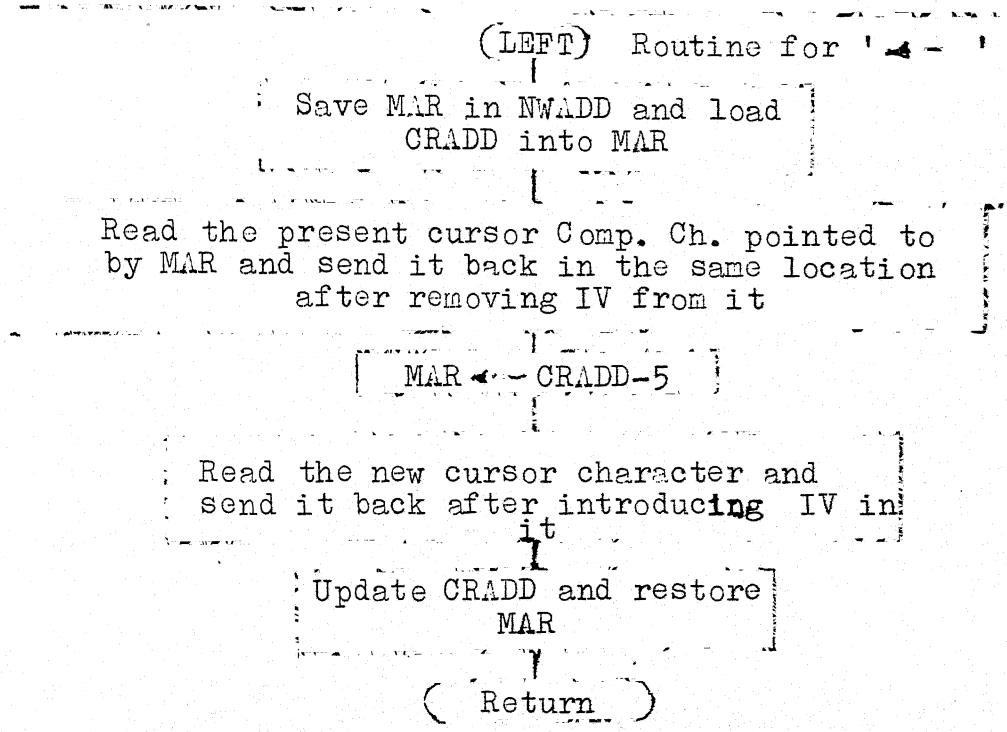


Fig. 4.6: Routine for 'Cursor move left'

are taken by the routine to do this:

1. Present contents of display MAR are saved in NWADD and contents of CRADD are in turn loaded into the MAR.
2. Inverse video (IV) bits are removed from the previous cursor position (pointed to by MAR).
3. IV bits are introduced in each of the constituent symbols of the next composite character.
4. Update CRADD and restore MAR.

Deviations from this simple routine occure for the following cases:

(i) When the previous cursor character is the last character in the page. To move the cursor to the right the system controller now fills a blank composite character in the next (five) locations and introduces IVs in it.

(ii) When the previous cursor character is the last character in the line but not the last in the page. The system controller still has to fill an IVed blank composite character in the next five locations but now the memory contents have to be shifted too by five words, to create the necessary gap before the blank C.C. can be introduced there.

The flow chart is shown in Fig. 4.7.

4.2.4.4 Cursor_move_left:

This routine [Fig. 4.6] when called moves the cursor left by one composite character.

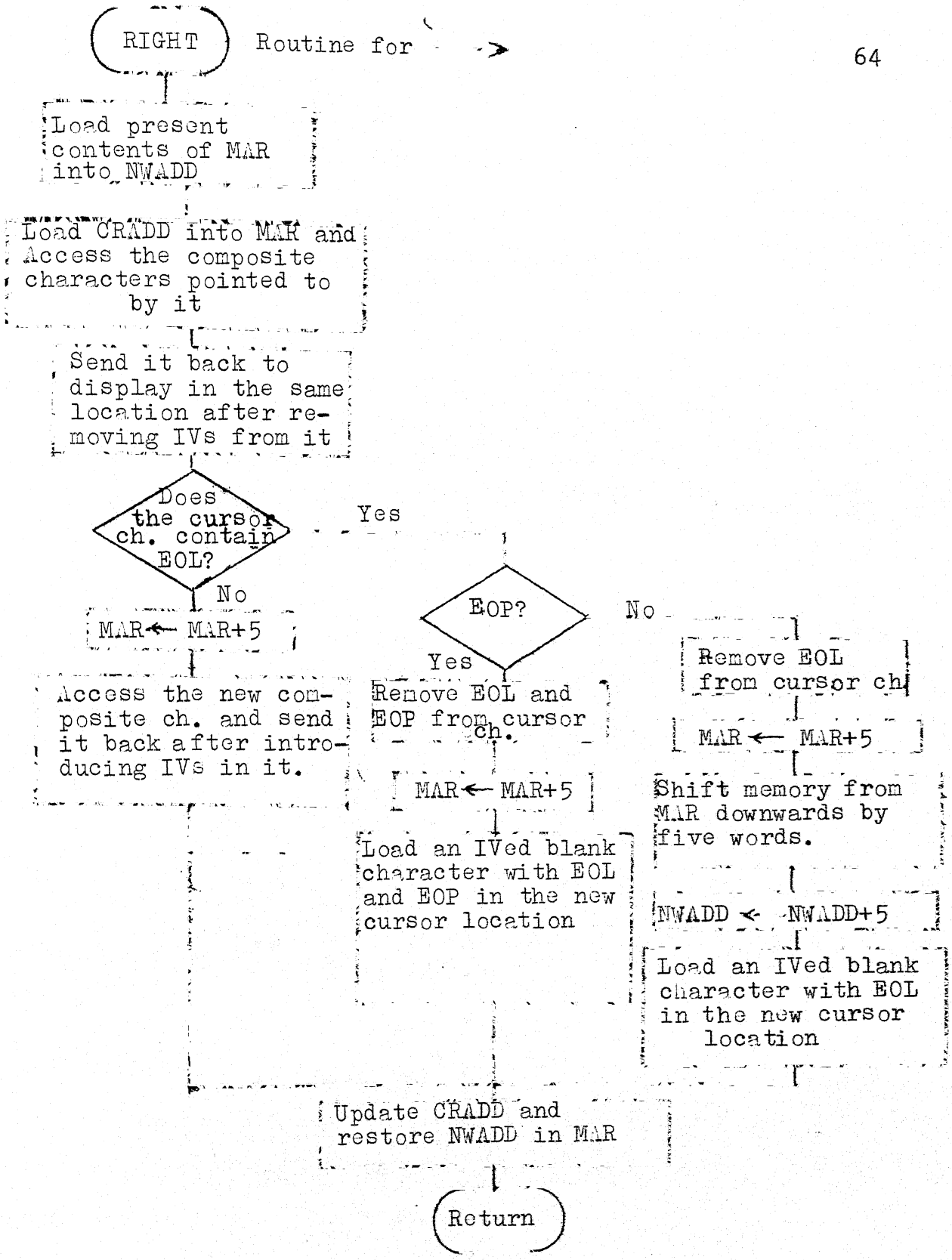


Fig. 4.7: Routine for 'cursor move right'

4.2.4.5 Cursor move down (↓):

Whatever the present cursor position, this routine moves it to the first composite character in the next line. Following steps are taken by the routine to do this:

1. Save present contents of MAR in NVADD and load it with CRADD.
2. Remove IV from present cursor character.
3. Starting from CRADD scan the display memory in dforward direction for EOL.
4. Move to next composite character and introduce IV in it.
5. Update CRADD and restore MAR.

Deviations from this routine occure for the following cases:

(i) When the line carrying cursor character is the last line in the text i.e. EOL is accompained by EOP.

(ii) When the line carrying cursor character is the last line entered so far on the screen, i.e. there are no composite characters in the next line nor does the present line contain EOP. In both cases the routine fills an IVed blank character in the first character position of the next line.

The flow chart is given in Fig. 4.8.

4.2.4.6 Cursor move up (↑):

Whatever the present position of the cursor, the routine [Fig. 4.9] always moves it to the first character of the previous line in the text.

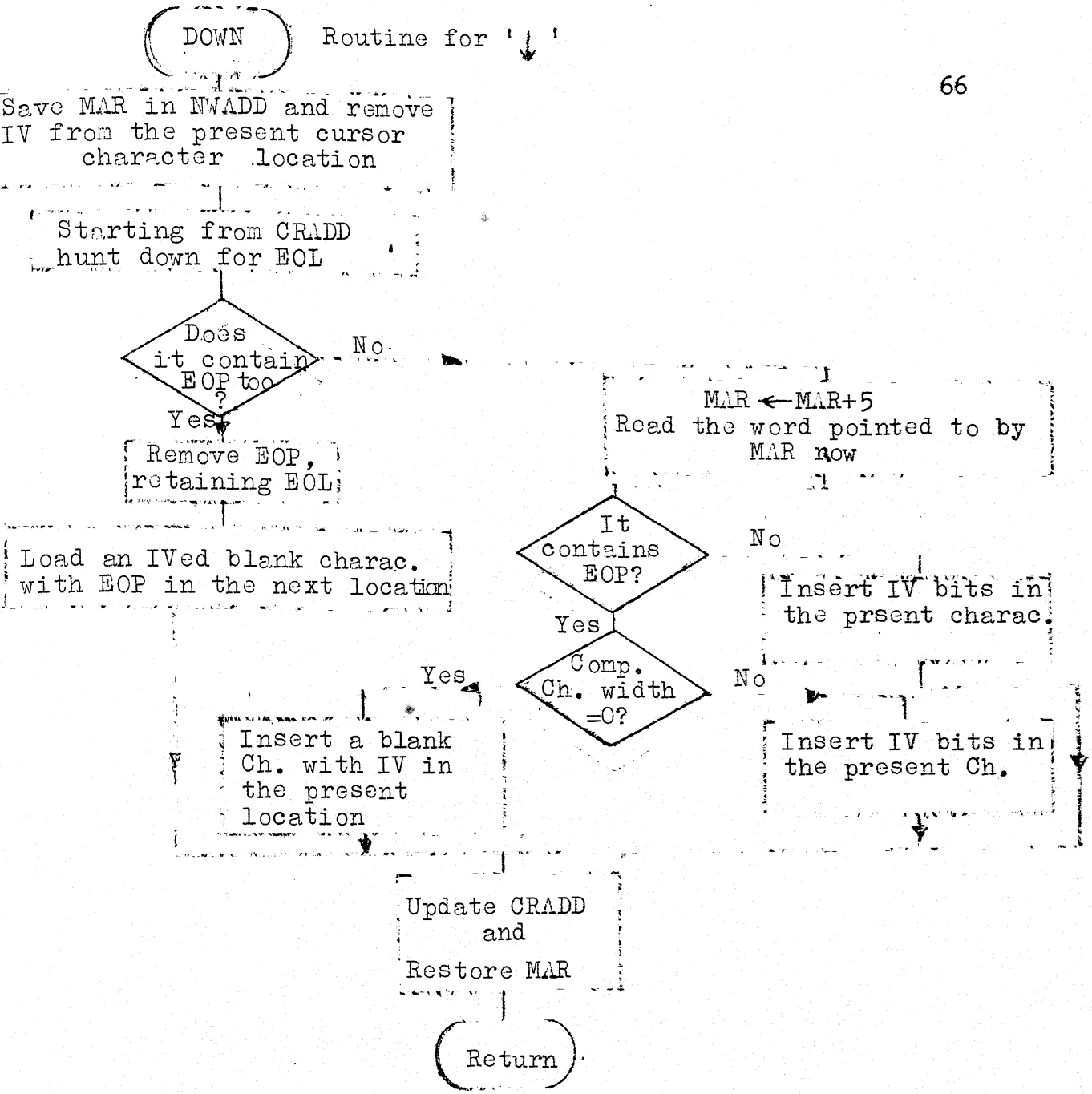


Fig. 4.8: Routine for Cursor move down

4.2.4.7 Display clear:

This routine clears the display and tones up the system for receiving the next page of the text from KB. This routine is the same as the intialization routine.

4.2.4.8 GO:

This routine just loads the contents of CRADD into MAR so that subsequent characters from KB get entered from the cursor position onwards. This facility is useful in giving headings, making paragraphs etc.

4.2.4.9 Character delete:

This routine when called erases as well as compresses the composite character marked by the cursor by filling zeros in the locations occupied by the character. The EOL and EOP bits of the character are however retained. The CRADD is left undisturbed.

4.2.4.10 Line delete:

The whole line containing the cursor character is erased as well as compressed. CRADD is moved to the erased line.

Flow chart for line delete is given in Fig. 4.11.

4.2.4.11 Character insert:

This routine creates a gap of five words at the cursor location (pointed to by CRADD) and loads CRADD into MAR in preparation for the character insert. The composite character to be inserted can now be inputted through the KB. CRADD is

Fig. 4.9: Routine for moving the cursor up.

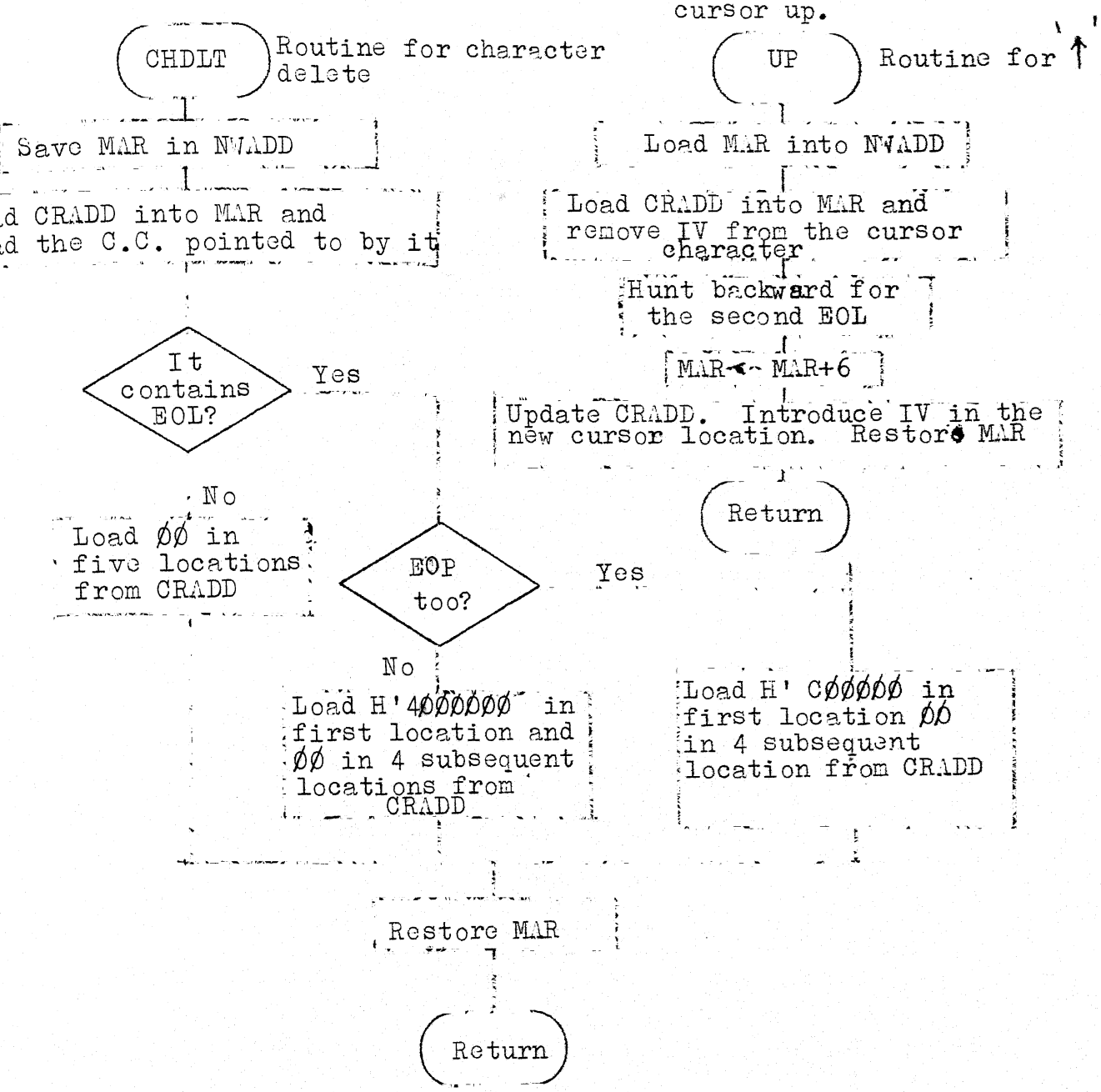


Fig. 4.10: Routine for character delete

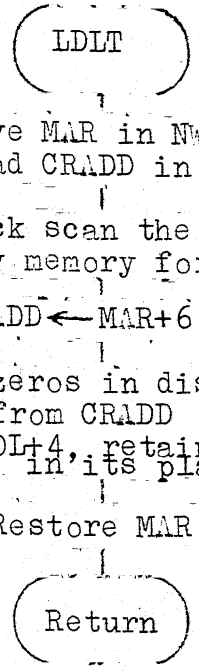


Fig. 4.11: Routine for line delete

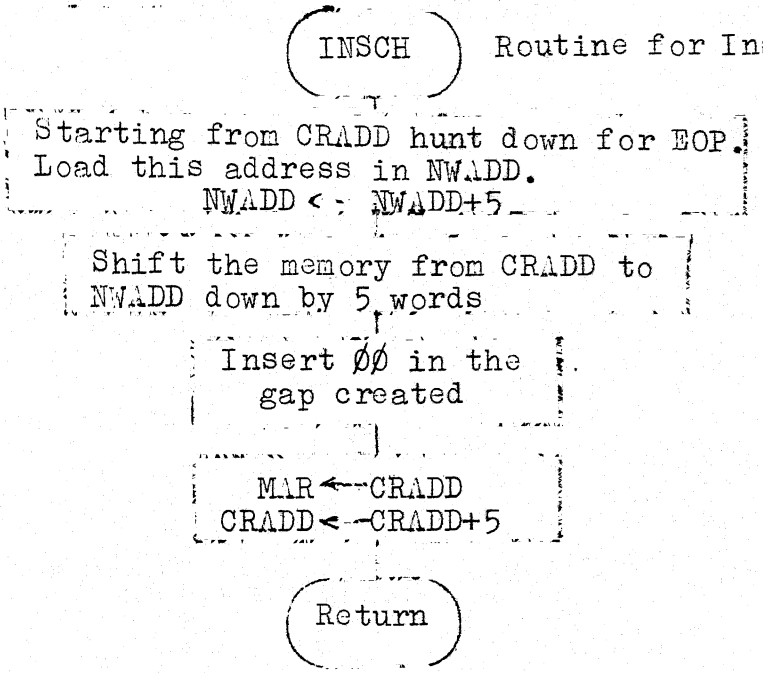


Fig. 4.12: Routine for character insert

incremented by 5.

The flow chart is given in Fig. 4.12.

4.3 Organisation of System Controller Routines

As is evident from the preceeding flow charts, some of the blocks are exceedingly repititive. These blocks have been separated into subroutines which are called by the main routines. (These subroutines may in turn call other subroutines). Some of the important subroutines are RBYTE, SBYTC, SBYTD (explained earlier), Store MAR in NWADD, Load CRADD/ NWADD into MAR, Read a composite character from display memory, Load a composite character into display memory, Introduce/Remove IV, Hunt for EOL etc. The contents of the subroutines have been trimmed for optimal total execution times and memory occupation of different routines. The routines themselves are implemented in the assembly language of 6800 and a listing of all subroutines and main programs is given in Apprendix C.

CHAPTER 5

CONCLUSION

The prime objective of this thesis was to evolve and test a strategy to be adopted for optimally developing the variety of peripherals for Indian scripts. Conditions for optimality include the need for developing modular peripherals which when linked together could be expanded into a number of possible configurations for Business Machines, Computer Terminals, Communications and Computer aided photo-type setting.

In this thesis, with this in view, the IEEE-488 standard Interface structure has been recommended and constructed for an Input/Output terminal being developed at IIT Kanpur. The Keyboarding scheme suggested by Narasimham, for Indian Scripts, has been implemented and the structure consisting of a Keyboard and System Controller has been interfaced to a CRT terminal and tested functionally.

Towards completion of this I/O terminal, the composition processor has to be connected to the Interface bus. This, however should poose no real problems as management and scheduling routines have been suggested for this purpose in this thesis.

REFERENCES

1. A. Raman, P.V.H.M.L. Narsimhan and R.M.K. Sinha, 'System modules for business machines, computer terminals and printing in Indian Languages'. Symposium on the use of Indian Languages in Computer based information systems, New Delhi, 1978.

2. P.V.H.M.L. Narsimham and A. Raman, 'A CRT based phototype setter for Indian Languages'. Symposium on the use of Indian Languages in computer based information systems, New Delhi, 1978.

3. R.M.K. Sinha, H.V. Sahasrabuddhe and V.K. Vaishnavi, 'Mechanization of Indian Scripts' CSI-1977, Poona.

4. P.V.H.M.L. Narsimham, B. Prasada, V. Rajaraman, 'Code based keyboard for Indian Languages' Journal of the Computer Society of India, Vol. 2 No. 2, Dec. 1971.

5. R.M.K. Sinha and H.N. Mahabala 'MODS Machine Oriented Devanagari Script' Journal of Institute of Telecommunication Engineers, Vol. 19, No. 11, Nov. 1973.

6. Y. Paker (Editor) 'Minicomputer Intefacing-course proceedings', Publisher Miniconsult Ltd., 1975.

7. Wakankar, L.S. 'Devanagari - Characteristics and Mechanization - a Collection of Essays'. The Commercial Printing Press Ltd., Bank Street Bombay, 1966.

8. IEEE Standard Digital Interface for Programmable Instrumentation (IEEE Std. 488-1975), April, 1975.

9. Donald C. Loughry, 'Interface standard 488 in action : Concepts and capabilities', ELECTRO'76.

10. A. Raman, A.K. Pathak, B.V. Ramana- Design Sheets for Fabricating the IEEE-488 Interface. ACES, IIT Kanpur, March 1978.

11. M.P. Sastri, 'A universal script generator for Indian Languages', M.Tech. thesis report, Deptt. of Electrical Engineering, May, 1978.

12. K.P. Laturkar, 'Devanagari script composition from phonetically coded symbols strings', M.Tech. thesis report, Computer Centre, IIT Kanpur.

13. 'Design an IEEE-488 bus into an FPLA' Electronic Design. Nov. 22, 1977.

14. Motorola 6800 Manual.

15. Hewlett Packard Journal, June and Jan. 1975.

16. A.K. Pathak, A. Raman, R.M.K. Sinha, 'A modular Indian Language Input/Output terminal'. Symposium on the use of Indian Languages in Computer based information systems, New Delhi, 1978.

17. Arjun Raman, M.P. Sastri, R.M.K. Sinha, 'A universal I/O device for Indian scripts'. Computer Society of India, Annual convention, Calcutta, March 1978.

18. David W. Ricci, Gerald E. Nelson 'Standard instrument interface simplifies design', Electronics Nov. 14, 1974.

19. Jane G. Evans and Howard Merrill, 'A standard interface applied to measurement systems', ISA proceeding of 22nd International Instrumentation Symposium, Vol. 3, 1976

APPENDIX A

USER'S MANUAL

After power on the first step in initializing the system is to program the interface functions. For this flick the Prog/Exec front panel switches of all IFR boxes to the Prog end. Now punch G, E711 on the TTY connected to system controller to call the programming routine in the system controller.

At the end of programming a PRG OVR is printed on the TTY by the routine.

Now flick all P/E switches back to Exec end and punch G, E21D - the start of system controller main routines.

An INIT OVR is printed on the TTY after the system has been initialized and is an indication to the operator that the system is ready for accepting data from the keyboard.

Some error messages may be printed on the TTY by the system controller during the system operation. They are INVLD INT when the system controller fails to find a requesting device during the serial poll routine; and ILLGL CCODE if the system controller receives an illegal control code from the KB. If these messages are the result of a key punch on the KB, the operator must punch the key again.

In case of a wrong entry during keyboarding, the cursor is brought to the position of the wrong entry and the entire composite character is deleted by punching 'delete character'

APPENDIX B

UNEXPLAINED ACRONYMS AND TERMS, USED IN THE TEXT

(i) Local messages: are messages between the interface functions of the device and its internal device functions.

(a) Local messages from device functions to the interface functions.

rdy - ready
nba - new byte available
pon - power on
tcs - take control synchronously
tca - take control asynchronously
rsv - request service
sic - send interface clear
rsc - request system control
gts - go to standby

(b) Local messages from interface functions to the device functions, like TACS, SPAS etc. are designer specified.

(ii) Remote messages: Remote messages are the messages sent over the interface bus via interface functions of two different devices. They are primarily meant to induce state transitions in the interface functions receiving them. They can be uniline (ATN, DAV etc), multiline (MLA, MTA etc.), device dependent (device data, status

byte etc.) and interface messages (device addresses, commands etc.)

MLA - My listen address

MTA - My talk address

UNL - unlisten

UNT - untalk

SPE - serial poll enable

SPD - serial poll disable

OTA - other's talk address

STB - status byte

Other Acronyms used in the text:

SMART - Screen Management Routines

PS - Present state (latch); Ref. fig. 2.3

NS - Next state (memory); Ref. fig. 2.3

CC - Condition code(Memory); Ref. fig. 2.3

IFR - Interface function repertcire (unit)

C.C. - Composite character

SBR - single byte memory read cycle

IV - Inverse video

MBR(R/W) - Memory Read/Write buffer register

EOL/EOP - End of line/page

MLAP - My programming listen address

MCAP - Memory capture signal Ref. fig. 2.7

CRADD - Cursor address

NWADD - Next word address

APPENDIX C
PROGRAM LISTING

PROGRAM

RAMMING ROUTINE

```
 86 FC N        LDAA#FC
 B7,FBC9        STAA FBC9
 B7 FBCB        STAA FBCB
 C6 FF          LDAB#FF      ALL PERIPHERAL LINES AS O/P'S
 BD E08E        JSR PRGPA
 BD E07D        JSR PRGPB
 CE E743        LDX#E743   LOAD STARTING ADD. OF DATA
 A6 00     LOOP LDAA 0,X
 E6 01          LDAB 1,X
 B7 FBC8        STAA FBC8
F7 FBCA        STAB FBCA
 C4 7F         LDAA#F4
 B7 FBCB        STAA FBCB
8E FC          LDAA#FC
 B7 FBCB        STAA FBCB   PULSE R/W LINE LO
 08             INX
 08             INX
 8C E79B        CMX#E79B  ALL DATA O/P'ED
26 E5         BNE,LOOP  NO? LOOP
 3F 12          SWI,LOOP   PRINT'PROG OVR'
 3F 80          SWI,80
```

VE P/E SWITCH TO 'P' MODE AND PUNCH G,E21D ON TTY

```
 BD E09F        JSR INIT
 CE E33B        LDX#E33B
 3F 12          SWI,PMSG    PRINT'INIT OVR'
 0E             CLI
 CE E235        LDX#E235    LOAD START OF INT ROUTINE
 FF FFF8        STX FFF8
 86 FD          LDAA#CC     ENABLE SRQ LINE(CB2)
 B7 FBCB        STAA FBCB
 3E             WAI
 7E E225        JMP E225
```

URTHER EXEC. STARTS FROM E235 ON OCCURANCE OF INT ON SRQ LINE

```
 C6 81         LDAB#81
BD E07D        JSR PRGPB    PROG ATN,DAV AS O/P'S
 86 C0          LDAA#C0
 BD E000        JSR SBYTC    SEND UNL
 86 E7          LDAA#E7
 BD E000        JSR SBYTC    SEND SPE
```

APPENDIX C
PROGRAM LISTING

MAIN PROGRAM

PROGRAMMING ROUTINE

```
E711  86 FC N        LDAA#FC
E713  B7,FBC9        STAA FBC9
E716  B7 FBCB        STAA FBCB
E719  C6 FF          LDAB#FF      ALL PERIPHERAL LINES AS O/P'S
E71B  BD E08E        JSR PRGPA
E71E  BD E07D        JSR PRGPB
E721  CE E743        LDX#E743   LOAD STARTING ADD. OF DATA
E724  A6 00     LOOP LDAA 0,X
E726  E6 01          LDAB 1,X
E628  B7 FBC8        STAA FBC8
E72B F7 FBCA        STAB FBCA
E72E  C4 7F         LDAA#F4
M730  B7 FBCB        STAA FBCB
733   8E FC         LDAA#FC
E735  B7 FBCB        STAA FBCB   PULSE R/W LINE LO
E738  08             INX
E739  08             INX
E73A  8C E79B        CMX#E79B  ALL DATA O/P'ED
73D   26 E5        BNE,LOOP  NO? LOOP
E73F  3F 12          SWI,LOOP   PRINT'PROG OVR'
E741  3F 80          SWI,80
```

X MOVE P/E SWITCH TO 'P' MODE AND PUNCH G,E21D ON TTY

```
E21D  BD E09F        JSR INIT
E220  CE E33B        LDX#E33B
E223  3F 12          SWI,PMSG    PRINT'INIT OVR'
E225  0E             CLI
E226  CE E235        LDX#E235    LOAD START OF INT ROUTINE
E229  FF FFF8        STX FFF8
E22C  86 FD          LDAA#CC     ENABLE SRQ LINE(CB2)
E22E  B7 FBCB        STAA FBCB
E231  3E             WAI
E232  7E E225        JMP E225
```

X FURTHER EXEC. STARTS FROM E235 ON OCCURANCE OF INT ON SRQ LINE

```
E235  C6 81         LDAB#81
237   BD E07D       JSR PRGPB    PROG ATN,DAV AS O/P'S
E23A  86 C0          LDAA#C0
E03C  BD E000        JSR SBYTC    SEND UNL
E23F  86 E7          LDAA#E7
E241  BD E000        JSR SBYTC    SEND SPE
```

```
E244  86 21          LDAA#21
E246  BD E000        JSR SBYTC    SEND MTA OF KB
E249  BD E05E        JSR SNRC
E24C  BD E03D        JSR RBYTE    RECEIVE STB OF KB IN ACC A
E24F  85 40          BITA#40
E251  27 51          BEQ,SRVKB    RQS='LO'?GO SERVE KB
E253  BD E06D        JSR RSNC
256  86 24          LDAA#24
E258  BD E000        JSR SBYTC    SEND MTA OF CH. COMPOSER
E25B  BD E05E        JSR SNRC
E25E  BD E03D        JSR RBYTE    RECV. STB OF CH.COMPOSER
E261  85 40          BITA#40
E263  27 13          BEQ,SRVCC
E005 PCE E344        LDX#E344
E268  3F 12          SWI PMSG     PRINT 'INVLD INT'
E26A  BD E06D        JSR RSNC
E26D  86 E6          LDAA#E6
E26F  BD E000        JSR SBYTC    SEND SPD
E272  86 20          LDAA#20
E274  BD E000        JSR SBYTC    SEND UNT
E277  3B             RTI

E278  BD E06D SRVCC  JSR RSNC
E27B  86 E6          LDAA#E6
E27D  BD E000        JSR SBYTC    SEND SPD
E280  86 C2          LDAA#C2
E282  BD E000        JSR SBYTC    SEND MLA OF DISPLAY
E285  C6 80          LDAB#80
E287  BD E07D        JSR PRGPB
E28A  C6 00          LDAB#00
E28C  BD E08E        JSR PRGPA
E28F  86 89          LDAA#89
E291  B7 FBCA        STAA FBCA    REMOVE ATN TO ALLOW THE TWO DEVICES
                                  TO TALK
E294  B6 FBCA  LP1   LDAA FBCA
E297  85 10          BITA#10      EOI='LO'?
E299  26 F9          BNE,LP1       NO, LOOP
E29B  BD E06D        JSR RSNC
E29E  86 20          LDAA#20
E2A0  BD E000        JSR RSNC     SEND UNT
E2A3  3B             RTI

E2A4  36       SRVKB PSHA         (A CONTAINS STB)
E2A5  BD E06D        JSR RSNC
E2A8  86 E6          LDAA#E6
E2AA  BD E000        JSR SBYTC    SEND SPD
E2AD  32             PULA
E2AE  85 0F          BITA#0F      D0,D1,D2,D3=0 IN STB?

E2B0  27 23          BEQ,SCMGT    YES,GO TO SCREEN MNGT ROUTINES
E2B2  86 C4          LDAA#C4
E2B4  BD E000        JSR SBYTC    MLA OF CH.COMPOSER
E2B7  C6 80          LDAB#80
E2B9  BD E07D        JSR PRGPB
E2BC  5F             CLRB
E2BD  BD E08E        JSR PRGPB
```

```
E2C0  86 89          LDAA#89
E2C2  B7 FBCA        STAA FBCA     REMOVE ATN TO ALLOW TWO DEVICES
                                    TO COMMUNICATE
E2C5  B6 FBCA  LP2   LDAA FBCA
E2C8  85 04          BITA#04       NDAC='HI'?
E2CA  27 F9          BEQ,LP2
E2CC  BD E06D        JSR RSNC
E2CF  86 20          LDAA#20
E2D1  BD E000        JSR SBYTC     SEND UNT
E2D4  3B             RTI

E2D5  BD E05E SCMGT  JSR SNRC
E2D8  BD E03D        JSR RBYTE     RECV. CODE FROM KB
E2DB  36             PSHA
E2DC  BD E06D        JSR RSNC
E2DF  86 20          LDAA#20
E2E1  BD E000        JSR SBYTC
E2E4  86 89          LDAA#89
E2E6  B7 FBCA        STAA FBCA
E2E9  32             PULA
E2EA  81 58          CMPA#58       CODE FOR CLEAR?
E2EC  27 2C          BEQ,CLR1
E2EE  81 08          CMPA#08-      CODE FOR CR?
E2F0  27 2E          BEQ,CR1
E2F2  81 10          CMPA#10       EOP?
E2F4  27 2A          BEQ,CR1
E2F6  81 38          CMPA#38       RIGHT?
E2F8  27 29          BEQ,RIGHT1
E2FA  81 30          CMPA#30       LEFT?
E2FC  27 28          BEQ,LEFT1
E2FE  81 28          CMPA#28       UP?
E300  27 27          BEQ,UP1
E302  81 20          CMPA#20       DOWN?
E304  27 26          BEQ,DN1
E396  81 18          CMPA#18       GO?
E308  27 25          BEQ,G01
E30A  81 70          CMPA#70       CH.DELETE?
E30C  27 24          BEQ,CHDLT1
E30E  81 68          CMPA#68       LINE DELETE?
E310  27 23          BEQ,LDLT1
E312  81 78          CMPA#78       CH.INSERT
E314  27 22          BEQ,CHINS1
E316  CE E34E        LDX#E34E      NONE? PRINT 'ILLGL CCODE'
E319  3F 12          SWI PMSG
E31B  3B             RTI
E31C  BD E09F CLR1   JSR INIT
E31F  3B             RTI
E320  7E E35A CR1    JMP CREOP
E323  7E E3EA RIGHT1 JMP RIGHT
E326  7E E4D1 LEFT1  JMP LEFT
E329  7E E524 UP1    JMP UP
E32C  7E E5ED DN1    JMP DN
E32F  7E E64D G01    JMP GO
E332  7E E6AE CHDLT1 JMP CHDLT
E335  7E E743 LDLT1  JMP LDLT
E338  7E E654 CHINS1 JMP CHINS
```

```
DATA
E33B  49 4E 49 54 20 4F 56 52 04 49 5U 56 4C 44 20 49 4E 54 04
E34E  49 4C 4C 47 4C 20 43 43 4F 44 45 04
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
ROUTINE FOR CR/EØP
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E35A  B7 FC3C CREOP STAA FC3C
E35D  CE FC27       LDX#FC27
E360  BD E0F5       JSR STONW    STORE MAR INTO NWADD
E363  F6 FC28       LDAB FC28
E366  B6 FC27       LDAA FC27
E369  C6 05         SUBB#05      MAR-5
E36B  82 00         SBCA#00
E36D  B7 FC00       STAA FC00
E370  F7 FC01       STAB FC01
E373  36            PSHA         SAVE TWO SETS OF MAR-5
E374  37            PSHB
E375  02
E376  86 C2         LDAA#C2
E378  BD E000       JSR SBYTC    SEND MLA OF DISPLAY
E37B  86 89         LDAA#89
E37D  B7 FBCA       STAA FBCB
E380  32            PULA
E381  BD E021       JSR SBYTD    SEND MAR-5 TO MAR
E384  32            PULA
E385  BD E021       JSR SBYTD
E388  86 D2         LDAA#D2
E38A  BD E000       JSR SBYTC    SEND MLAP
E38D  86 89         LDAA#89
E38F  B7 FBCA       STAA FBCA
E392  86 50         LDAA#50
E394  BD E021       JSR SBYTD    SEND 0101 ON D7,D6 D5 D4
E397  86 22         LDAA#22
E399  BD E000       JSR SBYTC
E39C  BD E05E       JSR SNRC
E39F  BD E03D       JSR RBYTE  RECV. MS BYTE OF DISPLAY
E3A2  B7 FC29       STAA FC29
E3A5  BD E03D       JSR RBYTE   RECV. MIDDLE BYTE
E3A8  B7 FC2A       STAA FC2A
E3AB  BD E03D       JSR RBYTE   RECV.LS BYTE
E3AB  B7 FC2B       STAA FC2B
E3B1  BD E06D       JSR RSNC
E3B4  CE FC00       LDX#FC00
E3B7  BD E11A       JSR LCRNW   RESTORE MAR-5 IN MAR
E3BA  86 C2         LDAA#C2
E3BC  BD E000       JSR SBYTC
E3BF  86 89         LDAA#89
E3C1  B7 FBCA       STAA FBCA
E3C4  B6 FC29       LDAA FC29
E3C7  F6 FC3C       LDAB FC3C   RECOVER THE CODE FROM KB
E3CA  C1 08         CMPB#08     CODE FOR CR?
E3CC  27 04         BEQ,CR
E3CE  8A C0         ORAA#C0     MAKE EOP=1
E3D0  20 02         BRA,EOP
E3D2  8A 40      CR ORAA#40     MAKE EOL=1
```

```
E3D4   BD E021   EOP   JSR SBYTD    SEND MS BYTE TO DISPLAY
E3D7   B6 FC2A         LDAA FC2A
E3DA   BD E021         JSR SBYTD
E3DD   B6 FC2B         LDAA FC2B
E3E0   BD E021         JSR SBYTD
E3E3   CE FC27         LDX#FC27
E3E6   BD E11A         JSR LCRNW    LOAD NWADDINTO MAR
E3E9   3B              RTI

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
ROUTINE FOR RIGHT
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 PROG. REG. CONTAINS 0101

E3EA   CE FC27 RIGHT LDX#FC27
E3ED   BD E0F5       JSR STONW    STORE MAR IN NWADD
E3F0   CE FC0A       LDX#FC0A
E3F3   BD E11A       JSR LCRNW    LD. CRADD INTO MAR
E3F6   CE FC29       LDX#FC29
E3F9   BD E14D       JSR RDCRA    READ CURSOR CH.INTO FC29 FC37
E3FC   CE FC2C       LDX#FC2C
E3FF   02
E400   BDEE170       JSR REMIV    REM. IV FROM CRSR.CH.
E403   B6 FC29       LDAAFC29
E406   85 40         BITA#40      EOL=1?
E408   26 34         BNE,LAST
E40A   CE FC40       LDX#FC40
E40D   BD E14D       JSR RDCRA    LD. NEXT COMP.CH. INTO FC40-FC4E
E410   CE FC43       LDX#FC43
E413   BD E187       JSR INTIV
E416   CE FC0A       LDX#FC0A
E419   BD E11A       JSR LCRNW    LD.CRADD INTO MAR
E41C   CE FC29       LDX#FC29
E41F   BD E19E       JSR LDACC    LD.INTO MEM.THE PREVIOUS CRSR.CH.
E422   CE FC40       LDX#FC40
E425   BD E19E       JSR LDACC    LD. THE NEW CRSR. CH.
E428   CE VC27  BACK LDX#FC27
E42B   BD E11A       JSR LCRNW    RESTORE MAR
E42E   86 05         LDAA#05
E430   BB FC0B       ADDA FC0B    CRADD=CRADD+5
E433   B7 FC0B       STAA FC0B
E436   4F            CLRA
E437   B9 FC0A       ADCA FC0A
E43A   B7 FC0A       STAA FC0A
E43D   3B            RTI
E43E   36       LAST PSHA
E43F   84 3F         ANDA#3F
E441   B7 FC29       STAA FC29    REMOVE EOP/EOL FROM CRSR. CH.
E444   CE FC0A       LDX#FC0A
E447   BD E11A       JSR LCRNW
E44A   CE FC29       LDX#FC29
E44D   BD E19E       JSR LDACC
E450   32            PULA
```

```
E455  36 CA      BR  LDAA#CA
E457  BD E021        JSR SBYTD
E45A  4F         BRCH CLRA        SEND FIRST CH. OF BLANK CH. WITH EOF
E45B  BD E021        JSR SBYTD
E45E  36 03          LDAA#03
E460  BD E021        JSR SBYTD
E463  36 41          LDAA#41
E465  BD E021-       JSR SBYTD
E463 36 23          LDAA#23
E46A  BD E021        JSR SBYTD
E46D  4F             CLRA
E46E  BD E021        JSR SBYTD
E471  C6 03          LDAB#03
E473  37        LOOP PSHB
E474  4F             CLRA
E475  BD E021        JSR SBYTD
E478  4F             CLRA         SEND REMAINING 3 SYMBOLS OF BLANK CH.
E479  BD E021        JSR SBYTD
E47C  4F             CLRA
E47D  BD E021        JSR SBYTD
E480  33             PULB
E481  5A             DECB
E482  26 EF          BNE,LOOP
E484  20 A2          BRA,BACK
E486  BD E1B6  SFT   JSR SHIFT
E489  B6 FC28        LDAAFC28
E48C  8B 05          ADDA#05      NWADD=NWADD+1
E48E  B7 FC23        STAA FC23
E491  B6 FC27        LDAA FC27
E494  89 00          ADCA#00
E496  B7 FC27        STAA FC27
E499  B6 FC0B        LDAA FC0B
E49C  8B 05          ADDA#05      ADD 5 TO CRADD
E49E  B7 FC0B        STAA FC0B
E4A1  B6 FC0A        LDAA FC0A
E4A4  89 00          ADCA#00
E4A6  B7 FC0A        STAA FC0A
E4A9  CE FC0A        LDX#FC0A
E4AC  BD E11A        JSR LCRNW    LD. CRADD INTO MAR
E4AF  86 C2          LDAA#C2
E4B1  BD E000        JSR SBYTC
E4B4  86 89          LDAA#89
E4B6  B7 FBCA        STAA FBCA
E4B9  86 4A      A6  LDAA#4A      SEND FIRST BYTE OF BLANK CH.
E4BB  BD E021        JSR SBYTD
E4BE  B6 FC0B        LDAA FC0B
E4C1  80 05          SUBA#05
E4C3  B7 FC0B        LDAA FC0B
E4C6  B6 FC0B        LDAA FC0B
E4C9  82 00          SBCA#00
E4CB  B7 FBCA        STAA FBCB
E4CE  7E E45A        JMP BRCH
XXXXXXX.XXXXXXXXXXXXXXXXXXXXXXXXXXX
ROUTINE FOR 'LEFT'
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E4D1  CE FC27  LEFT LDX#FC27
```

```
E4D4  BD EØF5        JSR STONW
E4D7  CE FCØA        LDX FCØA    LOAD CRADD INTO NBAR
E4DA  BD E11A        JSR LCRWW
E4DD  CE FC29        LDX FC29    LOAD CURSOR CH. INTO FC27-FC37
E4EØ  BD E14D        JSR RDCRA
E4E3  CE FC2C        LDX FC2C
E4E6  BD E17Ø        JSR REMIV
E4E9  B6 FCØB        LDAA FCØB
E4EC  8Ø Ø5          SUBA Ø5     CRADD=CRADD-5
E4EE  B7 FCØB        STAA FCØB
E4F1  B6 FCØA        LDAA FCØA
E4F4  82 Ø5          SBCA ØØ
E4F6  B7 FCØA        STAA FCØA
E4F9  CE FCØA        LDX FCØA
E4FC  BD E11A        JSR LCRNW
E4FF  CE FC4Ø        LDX FC4Ø    READ THE NEW CURSOR CH.
E5Ø2  BD E14D        JSR RDCRA
E5Ø5  CE FC43        LDX FC43
E5Ø8  BD E187        JSR INTIV   INTRODUCE IV
E5ØB  CE FCØA        LDX FCØA    RELOAD CRADD
E5ØE  BD E11A        JSR LCRNW
E511  CE FC4Ø        LDX FC4Ø    LOAD NEW CURSOR CH. WITH IV
E514  BD E19E        JSR LDACC
E517  CE FC29        LDX FC29    LOAD PREVIOUS CURSOR CH. WITHOUT
E51A  BD E19E        JSR LDACC
E51D  CE FC27        LDX FC27
E52Ø  BD E11A        JSR LCRNW   RESTORE NWADD
E523  3B             RTI
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
ROUTINE FOR 'DOWN'
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
E524  CE FC27  DN    LDX FC27
E527  BD EØF5        JSR STONW   STORE NWADD
E52A  CE FCØA        LDX FCØA
E52D  BD E11A        JSR LCRNW   LOAD CRADD INTO MAR
E53Ø  CE FC29        LDX FC29
E533  BD E14D        JSR RDCRA   RECEIVE THE CURSOR CH. IN TEMP B
E536  CE FC2C        LDX FC2C    REMOVE IV
E539  BD E17Ø        JSR REMIV
E53C  B6 FC29        LDAA FC29
E53F  85 4Ø          BITA 4Ø     IT CONTAINS EOL?
E541  27 2Ø          BEQ, A1     NO, BRANCH TO A1
E543  85 8Ø          BITA 8Ø     EOP?
E545  Ø7             TPA
E546  36             PSHA
E547  27 Ø8          BEQ, A2     NO BRANCH TO A2,
E549  B6 FC29        LDAA FC29
E54C  84 7F          ANDA 7F     REMOVE EOP
E54E  B7 FC29        STAA FC29
E551  CE FCØA  A2    LDX FCØA    LOAD CR ADD INTO MAR
E554  BD E11A        JSR LCRNW
E557  CE FC29        LDX FC29
```

```
E55A  BD E19E        JSR LDACC   SEND CURSOR CH. BACK
E55D  32             PULA
E55E  Ø6             TAP
E55F  27 49          BEQ,A4
E561  2Ø 44          BRA,A3
E563  CE FCØA  A1    LDX FCØA
E566  BD E11A        JSR LCRNW   LOAD CRADD INTO MAR
E569  CE FC29        LDX FC29    SEND CURSOR CH. BACK
E56C  BD E19E        JSR LDACC
E56F  C6 DØ          LDAB DØ     HUNT DOWN FOR EOL
E571  BD E1EF        JSR HTEOL
E574  36             PSHA
E575  CE FCØA        LDX FCØA    STORE MAR IN CRADD
E578  BD EØF5        JSR STONW
E57B  FE FCØA        LDX FCØA
E57E  Ø9             DEC
E57F  FF FCØA        STX FCØA
E582  32             PULA
E583  85 8Ø          BITA 8Ø     IT CONTAINS EOP TOO?
E585  27 23          BEQ, A4     NO, BRANCH TO A5
E587  CE FCØA        LDX FCØA
E58A  BD E11A        JSR LCRNW   LOAD CRADD INTO MAR
E58D  CE FC29        LDX FC29
E59Ø  BD E14D        JSR RDCRA
E593  B6 FC29        LDAA FC29
E596  84 7F          ANDA 7F     REMOVE EOP
E598  B7 FC29        STAA FC29
E59B  CE FCØA        LDX FCØA
E59E  BD E11A        JSR LCRNW
E5A1  CE FC29        LDX FC29
E5A4  BD E19E        JSR LDACC   LOAD COMP. CH. WITHOUT EOP INTO
E5A7  7E E455  A3    JMP, BR     ('RIGHT' ROUTINE)
E5AA  B6 FCØB  A4    LDAAFCØB
E5AD  8B Ø5          ADDA Ø5
E5AF  B7 FCØB        STAA FCØB   CRADD=CRADD+5
E5B2  B6 FCØA        LDAA FCØA
E5B5  89 ØØ          ADCA ØØ
E5B7  B7 FCØA        STAA FCØA
E5BA  CE FCØA        LDX FCØA
E5BD  BD E11A        JSR LCRNW
E5CØ  CE FC29        LDX FC29    ACCESS NEW CURSOR CH.
E5C3  BD E14D        JSR RDCRA
E5C6  B6 FC29        LDAA FC29
E5C9  85 8Ø          BITA 8Ø     IT CONTAINS EOP?
E5CB  27 Ø4          BEQ, A5     NO, BRANCH TO A5
E5CD  85 3F          BITA 3F     COMP CH. WIDTH=0?
D5CF  27 19          BEQ, A7     YES, BRANCH TO A6
E5D1  CE FCØA A5     LDX FCØA
E5D4  BD E11A        JSR LCRNW   NBAR   CRADD
E5D7  CE FC2C        LDX FC2C
E5DA  BD E187        JSR INSIV   INTRODUCE IV
E5DD  CE FC29        LDX FC29
E5EØ  BD E19E        JSR LDACC   SEND BACK THE NEW CURSOR CH.
E5E3  CE FC27        LDX FC27
E5E6  BD E11A        JSR LCRNW   RESTORE NWADD INTO MAR
E5E9  3B             RTI
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
ROUTINE FOR 'UP'
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

E5ED  CE FC27  UP    LDX FC27
E5FØ  BD EØF5        JSR STONW    STORE MAR INTO NWADD
E5F3  CE FCØA        LDX FCØA     LOAD CRADD INTO MAR
E5F6  BD E11A        JSR LCRNW
E5F9  CE FC29        LDX FC29
E5FC  BD E14D        JSR RDCRA    LOAD THE CURSOR CH. INTO TEMP
E5FF  CE FC2C        LDX FC2C     REMOVE IV
E6Ø2  BD E17Ø        JSR REMIV
E6Ø5  CE FCØA        LDX FCØA
E6Ø8  BD E17Ø        JSR LCRNW    LOAD CRADD AGAIN INTO MAR
E6ØB  CE FC29        LDX FC29
E6ØE  BD E19E        JSR LDACC    SEND BACK THE CURSOR CH. WITH IV
E661  C6 CØ          LDAB CØ                   RECEIVED
E613  BD E1EF        JSR HTEOL    HUNT FOR EOL BACKWARD
E616  C6 CØ          LDAB CØ
E618  BD E1EF        JSR HTEOL    HUNT BACKWARD FOR SECOND EOL
E61B  CE FCØA        LDX FCØA
E61E  BD EØF5        JSR STONW    LOAD MAR INTO CRADD
E621  B6 FCØB        LDAA FCØB
E624  8B Ø6          ADDA Ø6
E626  B7 FCØB        STAA FCØB    UPDATE CRADD
E629  B6 FCØA        LDAA FCØA
E62C  89 ØØ          ADCA ØØ
E62F  B7 FCØA        STAA FCØA
E631  BD E11A        JSR LCRNW    LOAD  CRADD TO MAR
E634  CE FC29        LEX FC29
E637  BD E14D        JSR RDCRA    READ THE NEW CURSOR CH.
E63A  CE FC2C        LDX FC2C     INSERT IV
E63D  BD E187        JSR INSIV
E64Ø  CE FC29        LDX FC29
E643  BD E19E        JSR LDACC    SEND BACK THE C.CH.
E646  CE FC27        LDX FC27     LOAD NWADD BACK INTO MAR
E649  BD E11A        JSR LCRNW
E64C  3B             RTT
E64D  CE FCØA  GO    LDX FCØA
E65Ø  BD E11A        JSR LCRNW    LOAD CRADD INTO MAR
E653  3B             RTI
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
ROUTINE FOR INSERT CH. (INSCH)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E654  CE FCØA INSCH LDX FCØA
E657  BD F11A        JSR LCRNW    LOAD CRADD INTO MAR
E65A  C6 DØ    LP    LDAB DØ
E65C  BD E1EF        JSR HTEOL    HUNT FORWARD FOR EOL
E65F  85 8Ø          BITA ØØ      EOP?
E661  27 F7          BEQ, LP
E663  CE FC27        LDX FC27
E666  BD EØF5        JSR STONW
E669  B6 FC28        LDAA FC28
E66C  8B Ø4          ADDA Ø4
E66E  B7 FC28        STAA FC28
```

```
E671  B6 FC27        LDAA FC27
E674  89 ØØ          ADCA ØØ
E676  B7 FC27        STAA FC27
E679  BD E1B6        JSR SHIFT
E67C  CE FCØA        LDX FCØA
E67F  BD E11A        JSR LCRNW
E682  86 C2          LDAA C2
E684  BD EØØØ        JSR SBYTC    SEND MLA OF DISPLAY
E687  86 89          LDAA 89
E689  B7 EBCA        STAA FBCA    ROMOVE ATN
E68C  C6 ØF          LDAB ØF
E68E  37      LOOP   PSHB
E68F  4F             LDAA ØØ      LOAD ØØ IN SURSOR CH. POSITION FOR
E 9Ø  BD EØ21        JSR SBYTD                          COMPRESSING
E 93  33             PULB
E695  5A             DECB
E695  26 F7          BNE, LOOP
E697  CE FCØA        LDX FCØA     LOAD CRADD INTO NBAR
E69A  BD E11A        JSR LCRNN
E69D  B6 FCØB        LDAA FCØB
E6AØ  8B Ø5          ADDA Ø5      CRADD=CRADD+5
E6A2  B7 FCØB        STAA FCØB    CRADD=CRADD+5
E6A5  B6 FCØA        LDAA FCØA
E6A8  89 ØØ          ADCA ØØ
E6AA  B7 FCØA        STAA FCØA
F6AD  3B             RTI
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
ROUTINE FOR 'CHARACTER DELETE'
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E6AE  CE FC27 CHDLT  LDX FC27
E6B1  BD EØF5        JSR STONW
E6B4  CE FCØA        LDX FCØA     LOAD CRADD INTO NBAR
E6B7  BD E11A        JSR LCRNW
E6BA  86 C2          LDAA C2
E6BC  BD EØØØ        JSR SBYTC
E6BF  86 89          LDAA 89
E6C1  B7 FBCA        STAA FBCA
E6C4  86 22          LDAA 22
E6C6  BD EØØØ        JSR SBYTC    SEND MTA OF DISPLAY
E6C9  BD EØ5E        JSR SNRC
E6CC  BD EØ3D        JSR RBYTE    RECEIVE MS BYTE OF CURSOR CH.
E6CF  36             PSHA
E6DØ  BD EØ3D        JSR RBYTE    TO BRING THE COUNT OF 3
E6D3  BD EØ3D        JSR RBYTE
E6D6  BD EØ6D        JSR RSNC
E6D9  32             PULA
E6DA  85 4Ø          BITA 4Ø      EOL=1?
E6DC  27 12          BEQ, B1      NO, BBRANCH TO B1
E6DE  85 8Ø          BITA 8Ø      EOP=1 TOO?
E6EØ  27 Ø7          BEQ,B/       NO, BRANCH TO B2
E6E2  86 CØ          LDAA CØ      MS BYTE OF ERASED CH.=Ø8 EXCEPT
E6E4  B7 FC29        STAA FC29
E6E7  2Ø ØA          BRA, B3
E6E9  86 4Ø     B2   LDAA 4Ø
E6EB  B7 FC29        STAA FC29    MS BYTE=ØØ EXCEPT EOL=1
E6EE  2Ø Ø3          BRA B3
```

```
E6F3  C6 ØE    B3     LDAB ØE
E6F5  CE FC2A         LDX FC2A
E6F8  6F ØØ    LOOP   CLR 0,X
E6FA  Ø8              INX          LOAD ØØ IN THE LOCATIONS FROM FC2A
E6FB  5A              DECB
E6FC  26 FA           BNE, LOOP
E6FE  CE FCØA         LDX FCØA     CRADD  NBAR
E7Ø1  BD E11A         JSR LCRNW
E7Ø4  CE FC29         LDX FC29
E7Ø7  BD E19E         JSR LDACC    LOAD ØØ (WITH EOL  EOP ETC) INTO THE
E7ØA  CE FC27         LDX FC27                    ERASED CH. POSITION
E7ØD  BD E11A         JSR LCRNW    RESTORE NWADD
E71Ø  3B              RTI
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
ROUTINE FOR 'LINE DELETE'
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E7AB  CE FC27  LDLT  LDX FC27
E7A6  BD EØF5        JSR STONW    STORE NWADD
E7A9  CE FCØA        LDX FCØA     LOAD CRADD INTO MAR
E7AC  BD E11A        JSR LCRNW
E7AF  CE CØ          LDAB CØ      BACK SCAN FOR EOL
E7B1  BD E1EF        JSR HTEOL
E7B4  CE FCØA        LDX FCØA     STORE MAR INTO CRADD
E7B7  BD EØF5        JSR STONW
E7BA  B6 FCØB        LDAA FCØB
E7BD  8B Ø6          ADDA Ø6      CRADD=CRADD+5
E7ØF  B7 FCØB        LTAA FCØB
E7C2  B6 FCØA        LDAA FCØA
E7C5  89 ØØ          ADCA ØØ
E7C7  B7 FCØA        STAA FCØA
E7CA  CE FCØA        LDX FCØA
E7CD  BD E11A        JSR LCRNW    LOAD CRADD INTO NBAR
E7DØ  C6 DØ          LDAB DØ      HUNT FORWARD FOR EOL
E7D2  BD E1EF        JSR HTEOL
E7D5  CE FC5Ø        LDX FC5Ø
D7D8  BD EØF5        JSR STØNW    MOVE MAR TO FC5Ø/51
E7DB  FE FC5Ø        LDX FC5Ø
E7DE  Ø9             DEX          MAR=MAR-1
E7DF  FF FC5Ø        STX FC5Ø
E7E2  4F             LDAA ØØ
E7E3  CE FC29        LDX FC29
E7E6  A7 ØØ    LOOP  STAA 0,X
E7E8  Ø8             INX          FILL ØØ IN FC29 TO FC37
E7E9  8C FC38        CMX FC38
E7EC  26 F8          BNE, LOOP
E7EE  CE FCØA        LDX FCØA     LOAD CRADD INTO MAR
E7F1  BD E11A        JSR LCRNW
E7F4  CE FC29 LDMOR  LDX FC29
FDF7  BD E19E        JSR LDACC
FDFA  FE FC5Ø        LDX FC5Ø     MAR=CRADD?
FDFD  BC FCØA        CMX FCØA
FDØØ  27 14          BEQ, LDOVR
FDØ2  Ø2             NOP
FDØ3  B6 FC51        LDAA FC51
FDØ6  8Ø Ø5          SUBA Ø5
```

```
FDØ8  B7 FD51      STAA FC51   MAR=MAR-5
FDØB  B6 FC5Ø      LDAA FC50
FDØE  82 ØØ        SBCA ØØ
FD1Ø  B7 FC5Ø      STAA FC50
FD13  7E E7F4      JMP LDMOR
FD16  CE FC27      LDX FC27    RESTORE NWABD INTO MAR
FD19  BD E11A      JSR LCRNW
FD1C  3B           RTI
```

```
                    XXXXXXXXXXXXXXXX
                      SUBROUTINES
                    XXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
SEND BYTE AS CONTROLLER
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

BYTE TO BE SENT IS ASSUMED IN ACCA

E000  CE 09 SBYTC LDAB 09
E002  F7 FBCA     STAB FBCA   SEND ATN='LO'
E005  F6 FBCA LPI LDAB FBCA
E008  C4 02       ANDB 02   NRFD=HI?
E00A  27 F9       BEQ, LPI
E00C  B7 FBC8     STAA FBC8   LOAD DATA ON BUS
E00F  86 08       LDAA 08
E011  B7 FBCA     STAA FBCA   SEND DAV='LO'
E014  B6 FBCA LP2 LDAA FBCA
E017  84 04       ANDA 04   NDAC=HI
E019  27 F9       BEQ, LP2
E01B  86 09       LDAA 09    REMOVE DAV
E01D  B7 FBCA     STAA FBCA
E020  39          RTS
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
SEND BYTE AS DEVICE
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E021  F6 FBCA LPI LDAB FBCA
E024  B4 02       ANDB 02   NRFO=HI?
E026  27 F9       BEQ, LPI
E028  B7 FBC8     STAA FBC8   LOAD DATA ON BUS
E02B  86 88       LDAA 88
E02D  B7 FBCA     STAA FBCA   SEND DAV=LO
E030  B6 FBCA LP2 LDAA FBCA
E033  84 04       ANDA 04   NDAC=HI?
E035  27 F9       BEQ, LP2
E037  86 89       LDAA 89   REMOVE DAV
E039  B7 FBCA     STAA FBCA
E03C  39          RTS
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
RECEIVE BYTE AS DEVICE
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E03D  86 8A RBYTE LDAA 8A   SEND NRFD-HI (NDAC='LO'+ATN=HI)
E03F  B7 FBCA     STAA FBCA
E042  B6 FBCA LPI LDAA FBCA
E045  84 01       ANDA 01   DAV=LO?
E047  26 F9       BNE, LPI
E049  B6 FBC8     LDAA FBC8   RECEIVE BYTE IN ACCA
E04C  C6 8C       LDAB 8C
E04E  F7 FBCA     STAB FBCA   SEND NDAC=HI, NRFD=LO
E051  F6 FBCA LP2 LDAB FBCA
E054  C4 01       ANDB 01   DAV=HI?
E056  27 F9       BEQ, LP2
E058  C6 89       LDAB 89   SEND NDAC, NRFD='LO'
E05A  F7 FBCA     STAB FBCA
E05D  39          RTS
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
SEND TO RECEIVE CONVERTER
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
RECEIVE
E05F  C6 86 SNRC  LDAB 86
E060  BD E07D     JSR PRGPB   PROG ATN, NDAC, NRFD AS IO/P'S
E063  5F          CLRB
E064  BD E08E     JSR PRGPA   PA0-PA7 AS INPUTS
E067  86 89       LDAA 89   REMOVE ATN
E069  B7 FBCA     STAA FBCA
E06C  39          RTS
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
RECEIVE TO SEND AS CONTROLLER CONVERTER
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E06D  86 09 RSNC  LDAA 09
E06F  B7 FBCA     STAA FBCA   SEND ATN=LO
E072  C6 81       LDAB 81
E074  BD E07D     JSR PRGPB   PROG ATN, DAV AS O/P.
E077  C6 FF       LDAB FF
E079  BD E08E     JSR PRGPA   PA0-PA7 AS O/P'S
E07C  39          RTS
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
PROGRAM PERIPHERAL LINES PB0-7 (PRGPB)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E07D  B6 FBCB PRGPB LDAA FBCB
E080  84 FB       ANDA FB   CRB2=0 FOR DDRB ACCESS
E082  B7 FBCB     STAA FBCB
E085  F7 FBCA     STAB FBCA   OUTPUT THE CONTENTS OF ACCB TO DDRB
E088  8A 04       ORAA 04
E08A  B7 FBCB     STAA FBCB   CRB2=1
E08D  39          RTS
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
PROGRAM PERIPHERAL LINES PA0-7 (PRGPA)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E08E  B6 FBC9 PRGPA LDAA FBC9
E091  84 FB       ANDA FB   CRA2=0 FOR DDRA ACCESS
E093  B7 FBC9     STAA FBC9
E096  F7 FBC8     STAB FBC8   OUTPUT THE CONTENTS OF ACCB TO DDRA
E099  8A 04       ORAA 04
E09B  B7 FBC9     STAA FBC9   CRA2=1
E09E  39          RTS

NOTE: BOTH THE ABOVE PROGRAMS ASSUME THE BUTE TO BE LOADED INTO
      RESPECTIVE DDR'S PRESENT IN ACCB.

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
INITIALIZATION ROUTINE (INIT)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E09F  C6 89  INTT LDAB 89   ATN, IFC AND DAV AS O/P AND OTHERS AS INP
E0A1  BD E07D     VSR PRGPB
E0A4  86 81       LDAA 81
E0A6  B7 FBCA     STAA FBCA   SEND IFC=LO
E0A9  86 0F       LDAA 0F
E0AB  4A      LPI DECA        LOOP FOR 92 µsec
E0AC  26 FD       BNE, LPI
E0AE  86 89       LDAA 89
```

```
E0B0   B7 FBCA        STAA FBCA   REMOVE IFC
E0B3   86 D2          LDAA D2     SEND MLA PROG OF DISPLAY
E0B5   BD E000        JSR SBYTC
E0B8   86 89          LDAA 89     REMOVE ATN
B0BA   B7 FBCA        STAA FBCA
B0BD   86 50          LDAA 50     SEND 0101 ON D7 D6 D5 D4
E0BF   BD E021        JSR SBYTD
E0C2   86 C2          LDAA C2
E0C4   BD E000        JSR SBYTC   SEND MLA OF DISPLAY
E0C7   86 89          LDAA 89     REMOVE ATN
E0C9   B7 FBCA        STAA FBCA
E0CC   CE 1000        LDX 1000    SET N=1000
E0CF   86 C0    LOOP  LDAA C0
E0D1   BD E021        JSR SBYTD   SEND ONE WORD TO DISPLAY MEM
E0D4   4F             CLRA        FOR INITIALIZATION
D0D5   BD E021        JSR SBYTD
D0D8   4F             CLRA
D0D9   BD E021        JSR SBYTD
D0DC   09             DEX         N=N-1
E0DD   26 F0          BNE, LOOP   N=0?  NO, LOOP
E0DF   86 81          DDAA 81     SEND IFC LO
E0E1   B7 FBCA        STAA FBCA
E0E4   86 0F          LDAA 0F     LOOP FOR 92 μsec
E0E6   4A       LP2   DECA
E0E7   26 FD          BNE, LP2
E0E9   86 89          LDAA 89     REMOVE IFC
E0EB   B7 FBCA        STAA FBCA
E0EE   7F FC0A        CLR FC0A    CLEAR CRADD
E0F1   7F FC0B        CLR FC0B
E0F4   39             RTS
```

NOTES: 1. THIS PROGRAM ASSUMES THE PROG REG TO BE CONTAINING ANYTHING AND LEAVES IT CONTAINING 0101.
2. ATN, IFC, DAV LINES ARE LEFT AS O/P'S OUTPUTTING HI, HI, HI.

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
STORE MAR IN NWADD/CRADD
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

X=FC27: STORE IN NWADD
X=FC0A: STORE IN CRADD

```
E0F5   86 D2    STONW LDAA D2
E0F7   BD E000        JSR SBYTC   SEND MLA PROG OF DISPLAY
E0FA   86 89          LDAA 89
E0FC   B7 FBCA        STAA FBCA   REMOVE ATN
E0FF   86 30          LDAA 30
E101   BD E021        JSR SBYTD   SEND 0011 ON D7 D6 D5 D4
E104   86 22          LDAA 22
E106   BD E000        JSR SBYTC   SEND MTA OF DISPLAY
E109   BD E05E        JSR SNRC
E10C   BD E03D        JSR RBYTE   RECEIVE LS 8 BITS OF MAR
E10F   A7 01          STAA 1,X  STORE IN NWADD/CRADD
E111   BD E03D        USR RBYTE   RECEIVE MS 4 BITS
E114   A7 00          STAA 0,X
```

```
E116  BD E06D        JSR RSNC
E119  39             RTS

      NOTES 1. PROG REG. IS LEFT CONTAINING 0011
            2. TALKER FUNCTION OF DISPLAY IS LEFT IN ACTIVE ST

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
LOAD CRADD/NWADD INTO MAR
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
THIS ROUTINE LOADS CRADD (X=FC0A) OR NWADD (X=RC27) INTO MAR

E11A  86 D2  LCRNW LDAA D2
E11C  BD E000      JSR SBYTC   SEND MLA PROG OF DISPLAY
E11F  86 89        LDAA 89
E121  B7 FBCA      STAA FBCA   REMOVE ATN
E124  86 30        LDAA 30
E126  BD E021      USR SBYTD   SEND 0011 ON D7 D6 D5 D4
E129  86 C2        LDAA C2
E12B  BD E000      JSR SBYTC   SEND MLA OF DISPLAY
E12E  86 89        LDAA 89
E130  B7 FBCA      STAA FBCA   REMOVE ATN
E133  A6 01        LDAA 1,X    SEND MS 4 BITS OF CRADD/NWADD
E135  BD E021      USR SBYTD
E138  A6 00        LDAA 0,X    SEND LS 5 BITS OF CRADD/NWADD
E13A  BD E021      JSR SBYTD
E13D  86 D2        LDAA D2
E13F  BD E000      JSR SBYTC   SEND MLA PROG
E142  86 89        LDAA 89
E144  B7 FBCA      STAA FBCA   REMOVE ATN
E147  86 50        LDAA 50
E149  BD E021      JSR SBYTD   SEND 0101 ON D7 D6 D5 D4
E14C  39           RTS
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
READ ONE COMPOSITE CH. FROM DISPLAY MEMORY
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

THE RECEIVED COMP. CHARACTER IS LEFT IN 15 LOCATIONS FOLLOWING
THE ADDRESS CONTAINED IN X EG. FOR X=FC29, the COMP. CH. IS ST
AS UNDER:
                PROG. REG. IS ASSUMED TO CONTAIN 0101

E14D  86 22 RDCRA  LDAA 22
E14F  BD E000      JSR SBYTC   SEND MTA OF DISPLAY
E152  BD E05E      JSR SNRC    RECEIVE MS BYTE OF CURSOR WORD
E155  86 0F        LDAA 0F     SET COUNTER N=15
E157  36     LOOP  PSHA
E158  BDE03D       JSR RBYTE
E15B  A7 00        STAA 0,X
E15D  08           INX
E15E  32           PULA
E15F  4A           DECA
E160  26 F5        BNE, LOOP
E162  BD E06D      JSR RSNC
E165  86 20        LDAA 20
E167  BD E000      JSR SBYTC   SEND UNT
E16A  86 89        LDAA 89
E16C  B7 FBCA      STAA FBCA   REMOVE ATN
E16F  39           RTS
```

```
XXXXXXXXXXXX
ROMOVE IV
XXXXXXXXXXXX
```

(IV IS REMOVED FROM 4 SYMBOL CH'S FIRST OF WHOM IS GIVE INDEX REG)

```
E17Ø  C6 BF  REMIV LDAB BF
E172  17           TBA
E173  A4 ØØ        ANDA 0,X
E175  A7 ØØ        STAA 0,X
E177  17           TBA
E178  A4 Ø3        ANDA 3,X
E17A  A7 Ø3        STAA 3,X
E17C  17           TBA
E17D  A4 Ø6        ANDA 6,X
E17F  A7 Ø6        STAA 6,X
E181  17           TBA
E182  A4 Ø9        ANDA 9,X
E184  A7 Ø9        STAA 9,X
E186  39           RTS
XXXXXXXXXXXXX
INTRODUCE IV
XXXXXXXXXXXXX
E187  C6 4Ø INTIV  LDAB 4Ø
E189  17           TBA
E18A  AA ØØ        ORAA 0,X
E18C  A7 ØØ        STAA 0,X
E18E  17           TBA
E18F  AA Ø3        ORAA 3,X
E191  A7 Ø3        STAA 3,X
E193  17           TBA
E194  AA Ø6        ORAA 6,X
E196  A7 Ø6        STAA 6,X
E198  17           TBA
E199  AA Ø9        ORAA 9,X
E19B  A7 Ø9        STAA 9,X
E19D  39           RTS
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
LOAD ONE COMP. CH. INTO DISPLAY MEMORY
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

E19E  86 C2  LDACC LDAA C2
E1AØ  BD EØØØ      JSR SBYTC    SEND MLA OF DISPLAY
E1A3  86 89        LDAA 89
E1A5  B7 FBCA      STAA FBCA    REMOVE ATN
E1A8  C6 ØF        LDAB ØF
E1AA  37    NXTBT  PSHB
E1AB  A6 ØØ        LDAA 0,X
E1AD  BD EØ21      JSR SBYTD    SEND MS BYTE OF COMP. CH.
E1BØ  Ø8           INX
E1B1  33           PULB
E1B2  5A           DECB   N=N-1
E1B3  26 F5        BNE, NXTBT     ALL 15 BYTES SENT?
E1B5  39           RTS
```

NOTE: THIS PROGRAM SENDS 15 BYTES TO DISPLAY - FIRST OF WH
BYTE IS POINTED TO BY X.

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
SHIFT MEMORY BY ONE COMP. CH. UPTO CRADD
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
E1B6  FE FC27 SHIFT   LDX FC27
E1B9  FF FC5Ø         STX FC5Ø    SAVE NWADD IN FC50/51
E1BC  BC FCØA RET     CMX FCØA    SHIFTING OVER?
E1BF  27 27           BLE, SFTOVR
E1C1  B6 FC28         LDAA PC28
E1C4  8Ø Ø5           SUBA Ø5
E1C6  B7 FC28         STAA FC28   NWADD=NWADD-5
E1C9  B6 FC27         LDAA FC27
E1CC  82 ØØ           SBCA ØØ
E1CE  B7 FC27         STAA FC27
E1D1  CE FC27         LDX FC27
E1D4  BD E11A         JSR LCRNW   LOAD NWADD INTO MAR
E1D7  CE FC29         LDX FC29
E1DA  BD E14D         JSR RDCRA   READ MEMORY COMP. CH. (
E1DD  CE FC29         LDX FC29 GETS INCREMENTED BY 5)
E1EO/ BD E19E         JSR LDACC   LOAD IT BACK INTO DISPL
E1E3  FE FC27         LDX FC27                      MEMORY
E1E6  2Ø D4           BRA RET
E1E8  FE FC50/SFTOVR  LDX FC5Ø    RECOVER ORIGINAL NWADD
E1EB  FF FC27         STX FC27
E1EE  39              RTS
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
HUNT FOR EOL (FORWARD OR BACKWARD)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    For Forward Hunting B=DØ, Backward Hunting B=CØ

E1EF  37      HTEOL   PSHB
E1FØ  86 D2           LDAA D2
E1F2  BD EØØØ         JSR SEYTO   SEND MLAP
E1F5  86 89           LDAA 89
E1F7  B7 FBCA         STAA FBCA   REMOVE ATN
E1FA  32              PULA
E1FB  BD EØ21         JSR SBYTD   SEND CONTENTS OF B INTO
E1FE  86 22           LDAA 22
E2ØØ  BD EØØØ         JSR SBYTC   SEND MTA OF DISPLAY
E2Ø3  BD EØ5E         JSR SNRC
E2Ø6  BD EØ3D LOOP    JSR RBYTE   RECEIVE MS BYTE OF MEMOR
E2Ø9  85 4Ø           BITA 4Ø   EOL=1?       IN CKENBL MODE
E2ØB  27 F9           BEQ, LOOP   NO, LOOP FOR MORE
E2ØD  36              PSHA
E2ØE  BD EØ6D         JSR RSNC
E211  86 2Ø           LDAA 2Ø
E213  BD EØØØ         JSR SBYTC   SEND UNT
E216  86 89           LDAA 89
E218  B7 FBCA         STAA FBCA   REMOVE ATN
E21B  32              PULA
E21C  39              RTS
```

A 55456

Date Slip A55456

This book is to be returned on the date last stamped.

CD 6.72.9

EE-1978-M-PAT-INP